http://inforum.org.pt

2ª EDIÇÃO

INFORUM

SIMPÓSIO DE INFORMÁTICA

9 E 10 DE SETEMBRO DE 2010

UNIVERSIDADE DO MINHO

EDITORES

Luís S. Barbosa

Miguel P. Correia

INForum 2010

Actas do II Simpósio de Informática

Universidade do Minho, 9-10 Setembro, 2010

Luís S. Barbosa, Miguel P. Correia (Eds.)

Universidade do Minho
9-10 Setembro, 2010

Conteúdo

Prefácio

O presente volume reune as actas do II Simpósio de Informática — INForum 2010, realizado em 9 e 10 de Setembro de 2010, na Universidade do Minho. Dando continuidade à experiência pioneira do ano anterior, de novo este Simpósio se afirmou como um *ponto de encontro* e de *partida*.

Ponto de *encontro* de investigadores, docentes e alunos de pós-graduação em Informática de todo o país. Encontro de pessoas e equipas, oportunidade de interacção, divulgação de resultados recentes e projectos de investigação.

Mas, talvez mais do que isso, ponto de *partida* na afirmação da maturidade desta área científica em Portugal e na sua concretização através de novas sinergias, da confluência de interesses, da afirmação de parcerias inovadoras, da interacção crescente entre as Academias e a Indústria.

O programa deste ano inclui 3 conferências convidadas, 60 artigos longos (correspondentes a uma taxa de aceitação de 44% sobre o volume de submissões) e 14 artigos curtos. Os artigos foram submetidos a uma das 12 áreas temáticas previstas:

- Ciência e Engenharia de Software
- Compiladores e Linguagens de Programação
- Computação Distribuída e de Larga Escala
- Computação Gráfica
- Computação Móvel e Ubíqua
- Engenharia Conduzida por Modelos
- Especificação, Verificação, e Teste de Sistemas Críticos
- Gestão e Tratamento de Informação
- Internet das Coisas e Serviços
- Segurança de Sistemas de Computadores e Comunicações
- Sistemas Embebidos e de Tempo-Real
- Sistemas Inteligentes

Cada uma destas áreas resultou de uma candidatura apresentada por um conjunto de investigadores de diferentes instituições e é dotada de uma Comissão Científica própria responsável pela definição da chamada de trabalhos e pelo processo de selecção.

A colaboração empenhada dos coordenadores de cada uma das áreas temáticas do INForum 2010, a quem publicamente agradecemos, permitiu chegar a um programa, simultaneamente plural e coerente. Mas permitiu também prosseguir o esforço de interacção e conciliação de modos diversos de fazer e avaliar investigação em diferentes

sub-domínios da Informática.

Estamos em crer que, mais uma vez, o resultado deste esforço é bem maior que a soma das suas partes, afirmando este Simpósio como um evento de referência no âmbito português, e com progressiva atractividade internacional.

O caminho, sabemo-lo bem, é o andar que o traça. Mas, por isso mesmo, a exigência com que o edificarmos será a mais segura garantia para o futuro de uma área de investigação de premente relevância social e na qual o país tem provas dadas.

Como coordenadores científicos do INForum 2010 é-nos grato reconhecer o apoio do presidente da Comissão Coordenadora, Prof. Rui Oliveira, dos coordenadores locais, Prof. António Nestor Ribeiro e Prof. Manuel Alcino Cunha, assim como de todos os colegas e alunos que de alguma forma, mas sobretudo com a sua presença e entusiasmo, ajudaram a erguer este evento.

Luís S. Barbosa, Miguel P. Correia
Setembro 2010

Organização

O INForum 2010 foi organizado pelo *Centro de Ciências e Tecnologias da Computação*, CCTC, na Universidade do Minho, nas instalações da qual decorreram todas as sessões.

Coordenação

Comissão de Programa:	Luís S. Barbosa (U. Minho) Miguel P. Correia (U. Lisboa)
Comissão Organizadora:	António Nestor Ribeiro (U. Minho) Manuel Alcino Cunha (U. Minho)
Coordenador Edição das Actas:	José João Almeida (U. Minho)
Comissão para a Imagem e Divulgação:	Dulce Domingos (U. Lisboa) Jorge Sousa Pinto (U. Minho)
Comissão Coordenadora:	Ademar Aguiar (U. Porto) Ana Moreira (U. N. Lisboa) António Casimiro (U. Lisboa) Eduardo Tovar (I. P. Porto) Fernando Lobo (U. Algarve) José Luís Oliveira (U. Aveiro) Luís Rodrigues (Instituto Superior Técnico) Marco Vieira (U. Coimbra) Rui Lopes (I. P. Bragança) Rui Oliveira (U. Minho), Presidente Simão Sousa (U. Beira Interior)
Webmaster:	José Luís Faria (U. Minho)

Comissão de Selecção do Prémio *"Melhor Artigo Redigido por Estudante"*

José Luiz Fiadeiro (U. Leicester)	Reino Unido
Pedro Trancoso (U. Cyprus)	Chipre
Rodrigo Rodrigues, (MPI-SWS)	Alemanha

Comissões de Programa por Área Temática

Ciência e Engenharia de Software

Salvador Abreu (U. Évora)
João Cachopo I(ST)
João Pascoal Faria (U. Porto)
Mário Florido (U. Porto)
Antónia Lopes (U. Lisboa)
Paulo Marques (U. Coimbra)
Ana Moreira (U. N. Lisboa)
Fernando Silva (U. Porto)
Vasco Vasconcelos (U. Lisboa)
Ademar Aguiar (U. Porto)
Luís Caires (U. N. Lisboa), *Coord.*
João M. Fernandes (U. Minho)
Lúcio Ferrão (OutSystems SA)
Inês Lynce (IST)
Paulo Mateus (IST)
José N. Oliveira (U. Minho)
Simão Melo de Sousa (U. Beira Interior)

Compiladores e Linguagens de Programação

Alberto Simões (I. P. Porto)
António Menezes Leitão (U. T. Lisboa)
Bostjan Slivnik (U. Ljubljana, Eslovénia)
Daniel Riesco (U. San Luis, Argentina)
German Montejano (U. San Luis, Argentina)
Ivan Lukovic (U. Novi Sad, Sérvia)
João M. P. Cardoso (U. Porto)
João Saraiva (U. Minho)
Manuel Pérez Cota (U. Vigo, Espanha)
Marjan Mernik (U. Maribor, Eslovénia)
Mario G. Leguizamón (U. San Luis, Argentina)
Nuno Rodrigues (I. P. Cávado e Ave)
Pedro R. Henriques (U. Minho), *Coord.*
Salvador Abreu (U. Évora)
Susan Esquivel (U. San Luis, Argentina)
Vasco Amaral (U. N. Lisboa)
Xavier Gómez Guinovart (U. Vigo, Espanha)
Alda Gançarski (I. T. M, SudParis, França)
Bastian Cramer (U. Paderborn, Alemanha)
Casiano Rodríguez León (U. La Laguna, Espanha)
Daniela da Cruz (U. Minho)
Giovani Librelotto (U. F. Santa Maria, Brasil)
Jean-Cristophe Filliâtre (U. Paris Sud 11, França)
João Costa Seco (U. N. Lisboa)
José João Almeida (U. Minho)
Maria João Varanda Pereira (I. P. Bragança)
Mario Berón (U. San Luis, Argentina)
Matej Crepinsek (U. Maribor, Eslovénia)
Paulo Matos (I. P. Bragança)
Rogério Dias Paulo (Efacec)
Simão Melo de Sousa (U. Beira Interior)
Tomaz Kosar (U. Maribor, Eslovénia)
Vitor Santos (Microsoft Portugal)

Computação Distribuída e de Larga Escala

Alysson Bessani (U. Lisboa)
António Sousa (U. Minho)
David Matos (IST)
Hugo Miranda (U. Lisboa)
João Lourenço (U. N. Lisboa)
Jorge Gomes (LIP)
José Vermelhudo (NAV)
Luís Moura e Silva (U. Coimbra)
Luís Rodrigues (IST)
Nuno Duro (Evolve Space Solutions)
Paula Prata (U. Beira Interior)
Paulo Marques (U. Coimbra)
Pedro Bizarro (U. Coimbra)
Przemyslaw Lenkiewicz (Microsoft Portugal)
Sérgio Duarte (U. N. Lisboa)
Antonio Luis Osório (I. S. E. Lisboa)
Bruno Schulz (Lab. Nac. Comp. Científica, Brasil)
Henrique J. Domingos (U. N. Lisboa)
João Barros (U. Porto)
João Paulo Carvalho (IST)
José Orlando Pereira (U. Minho)
Luís Miguel Pinho (ISEP - I. P. Porto)
Luís Oliveira e Silva (IST)
Luís Veiga (IST), *Coord.*
Paul Grace (U. Lancaster, Reino Unido)
Paulo Ferreira (IST)
Paulo Vilela (Sun Microsystems Portugal)
Pedro Furtado (U. Coimbra)
Renato Cerqueira (PUC-Rio, Brasil)

Computação Gráfica

Abel Gomes (U. Beira Interior)
Adriano Lopes (U. N. Lisboa)
Ana Paula Cláudio (U. Lisboa)
António Augusto Sousa (U. Porto)
António Ramires Fernandes (U. Minho)
Elisabeth Simão Carvalho (U. Minho)
Fernando Birra (U. N. Lisboa)
Frutuoso Silva (U. Beira Interior)
Hans du Buf (U. Algarve)
João Cunha (LNEC)
João Paulo Moura (UTAD)
Joaquim Madeira (U. Aveiro)
José Creissac Campos (U. Minho)
Luís Gonzaga Magalhães (UTAD), *Coord.*
Luís Paulo Santos (U. Minho), *Coord.*
Manuel João Fonseca (IST)
Maria Beatriz Carmo (U. Lisboa)
Maximino Bessa (UTAD)
Miguel Sales Dias (ISCTE, Microsoft)
Nuno Correia (U. N. Lisboa)
Paulo Dias (U. Aveiro)
Pedro Moreira (ESTG - I. P. Viana do Castelo)
Teresa Romão (U. N. Lisboa)
Adérito Marcos (CCG / U. Aberta)
Alberto Proença (U. Minho)
Antão Almada (YDreams)
Beatriz Sousa Santos (U. Aveiro)
João Paulo Carvalho (IST)
Fernando Nunes Ferreira (U. Porto)
Francisco Morgado (ESTV - I. P. Viseu)
Gonçalo Lopes (YDreams)
Ido Iurgel (CCG / U. Minho)
João Madeiras Pereira (IST)
Joaquim Jorge (IST)
José Carlos Teixeira (U. Coimbra)
Leonel Valbom (Escola Superior Gallaecia)
Luís Marcelino (ESTG - I. P. Leiria)
Manuel João Ferreira (U. Minho)
Manuel Próspero dos Santos (U. N. Lisboa)
Mário Rui Gomes (IST)
Miguel Leitão (ISEP - I. P. Porto)
Nelson Zagalo (U. Minho)
Nuno Jardim Nunes (U. Madeira)
Pedro Branco (U. Minho)
Teresa Chambel (U. Lisboa)
Vítor Santo (Microsoft Portugal)

Computação Móvel e Ubíqua

Adriano Moreira (U. Minho)
Carlos Baquero (U. Minho)
Daniel Gonçalves (IST)
Francisco Pereira (U. Coimbra)
Hugo Miranda (U. Lisboa), *Coord.*
Luís Veiga (IST)
Nuno Correia (U. N. Lisboa)
Nuno Preguiça (U. N. Lisboa)
Pedro Araújo (U. Beira Interior)
Rui José(U. Minho)
Ana Paula Afonso (U. Lisboa), *Coord.*
Carlos Bento (U. Coimbra)
Filipe Pacheco (ISEP - I. P. Porto)
Frederico Figueiredo (Nokia Siemens Networks)
Luís Carriço (U. Lisboa)
Manuel Sequeira (ZON)
Nuno Maria (Truewind)
Paulo Ferreira (IST)
Rui Andrade (NovaGeo)
Sérgio Duarte (U. N. Lisboa)

Engenharia Conduzida por Modelos

Alberto Rodrigues da Silva (IST), *Coord.*
António Leitão (IST)
David Ferreira (IST)
João Araújo (U. N. Lisboa)
João Paulo Carvalho (Quidgest)
João Saraiva (IST)
Leonel Nóbrega (U. Madeira)
Luís Pedro (D'Auriol Swiss)
Miguel Calejo (Declarativa)
Nuno Nunes (U. Madeira)
Ana Paiva (U. Porto)
Bruno Barroca (U. N. Lisboa)
Fernando Brito de Abreu (U. N. Lisboa)
João Miguel Fernandes (U. Minho)
João Pascoal Faria (U. Porto), *Coord.*
José Borbinha (IST)
Levi Lúcio (U. N. Lisboa)
Matteo Risoldi (U. Geneva)
Miguel Goulão (U. N. Lisboa)
Ricardo Machado (U. Minho)

Vasco Amaral (U. N. Lisboa), *Coord.*

Especificação, Verificação, e Teste de Sistemas Críticos

Ana Matos (U. T. Lisboa)
Ana Paiva (U. Porto)
Carla Ferreira (U. N. Lisboa)
Jorge Sousa Pinto (U. Minho), *Coord.*
José Miguel Faria (Critical Software)
Luís Miguel Pinho (ISEP - I. P. Porto)
Luís Pinto (U. Minho)
Mário Florido (U. Porto)
Nelma Moreira (U. Porto)
Nestor Catano (U. Madeira)
Sérgio Amado (EDISOFT)
Simão Melo de Sousa (U. Beira Interior)

Gestão e Tratamento de Informação

Alípio Jorge (U. Porto)
Bruno Martins (IST), *Coord.*
Daniel Gomes (FCCN)
David Matos (IST)
Diana Santos (SINTEF)
Eugénio de Oliveira (U. Porto)
Francisco Couto (U. Lisboa)
Gaël Dias (U. Beira Interior)
Helena Galhardas (IST), *Coord.*
Helena Sofia Pinto (IST)
Irene Rodrigues (U. Évora)
João Pereira (IST)
José João Dias de Almeida (U. Minho)
Luísa Coheur (IST), *Coord.*
Maribel Santos (U. Minho)
Mário Silva (U. Lisboa)
Nuno Cavalheiro Marques (U. N. Lisboa)
Nuno Mamede (IST)
Paulo Carreira (IST)
Paulo Gomes (U. Coimbra)
Paulo Jorge Oliveira (ISEP - I. P. Porto)
Paulo Quaresma (U. Évora)
Pavel Calado (IST)

Internet das Coisas e Serviços

António Rito-Silva (IST)
Caio Fontana (U. S. Paulo, Brasil)
Dulce Domingos (U. Lisboa), *Coord.*
Eduardo Dias (U. S. Paulo, Brasil)
Fabrício Silva (U. Lisboa)
Francisco Martins (U. Lisboa), *Coord.*
Henrique João Domingos (U. N. Lisboa)
Hervé Paulino (U. N. Lisboa)
João Campos (Truewind)
Jorge Cardoso (U. Coimbra)
Luis Lopes (U. Porto)
Ricardo Martinho (I. P. Leiria)

Segurança de Sistemas de Computadores e Comunicações

Alysson Bessani (U. Lisboa)
André Zúquete (U. Aveiro), *Coord.*
Carlos Ribeiro (IST), *Coord.*
Edmundo Monteiro (U. Coimbra)
Henrique Domingos (U. N. Lisboa), *Coord.*
José Alegria (Portugal Telecom)
Luis Antunes (U. Porto)
Marco Vieira (U. Coimbra)
Miguel Pupo Correia (U. Lisboa)
Nuno Neves (U. Lisboa)
Irene Rodrigues (U. Évora)
João Pereira (IST)
Paulo Ferreira (IST)
Paulo Simões (U. Coimbra)
Paulo Sousa (U. Lisboa)
Pedro Adão (IST)
Ricardo Chaves (IST)
Simão Melo de Sousa (U. Beira Interior)

Sistemas Embebidos e de Tempo-Real

Adelino Silva (LINCIS)
Carlos Almeida (IST)
Helder Silva (EDISOFT)
João Almeida (youmove, Lda)
João M. P. Cardoso (U. Porto)
João Cunha (I. S. E. Coimbra)

João Fernandes (U. Minho)
Joaquim Ferreira (U. Lisboa)
José Malaquias (ISA)
José Rufino (U. Lisboa), *Coord.*
Luís Almeida (U. Porto)
Luís Gomes (U. N. Lisboa)
Luís Osório (I. S. E. Lisboa)
Mike Rennie (DEIMOS Engenharia)
Nuno Pereira (ISEP - I. P. Porto)
Pedro Fonseca (Micro I/O)
Simão Melo de Sousa (U. Beira Interior)
João Redol (Nokia Siemens Networks)
José Fonseca (U. Aveiro)
José Metrôlho (EST-II. P. Castelo Branco)
Leonel Sousa (IST)
Luís Correia (EMPORDEF-TI)
Luís Miguel Pinho (ISEP - I. P. Porto), *Coord.*
Mário Calha (U. Lisboa)
Nelson Blanco (PDM&FC)
Nuno Silva (Critical Software S.A.)
Rui Camolino (Brisa, SA)
Tobias Schoofs (GMV-Skysoft)

Sistemas Inteligentes

Alberto Freitas (U. Porto)
César Analide (U. Minho)
José Machado(U. Minho), *Coord.*
Luis Antunes (U. Lisboa)
Manuel Santos (U. Minho)
Ricardo Oliveira (U. Porto)
Vitor Alves (U. Minho)
António Abelha (U. Minho), *Coord.*
Goreti Marreiros (ISEP - I. P. Porto)
José Neves (U. Minho), *Coord.*
Luis Moniz (U. Lisboa)
Paulo Novais (U. Minho), *Coord.*
Ricardo Santos (ESTGF - I. P. Porto)

Apoios

Apoio à Comissão de Progama
Banco Espírito Santo
Critical Software
Eurotux
EFACEC
Glintt HS
Hewlett-Packard
Microsoft
Seegno

Apoio organizativo:
Cafés Delta
LONA
Pastelarias Cristo-Rei

Apoio institucional
FCT - Fundação para a Ciência e Tecnologia
UM - Universidade do Minho
CISUC - Center for Informatics and Systems of the Univ. of Coimbra, U. Coimbra
CCTC - Centro de Ciências e Tecnologias da Computação, U. Minho
IEETA - Instituto de Engenharia Electrónica e Telemática de Aveiro, U. Aveiro
LASIGE - Large-Scale Informatics Systems Laboratory, U. Lisboa
CISTER - Research Centre in Real-Time Computing Systems, I. P. Porto

Conferências Convidadas

Distributed coordination

Resumo. Distributed coordination is critical for Web-scale distributed systems. Examples of such systems are distributed file systems used for data processing (e.g., GFS, HDFS1), scalable data storage systems (e.g., BigTable), and Web search infrastructure (e.g., Crawlers, Indexers). Coordination in these systems can take the form of configuration metadata or more sophisticated synchronization primitives such as locks and leader election. Some of these coordination needs arise frequently enough to justify the design and implementation of systems for coordination.

In this presentation, we motivate the design of systems to enable the co- ordination of distributed applications, and discuss the design of two systems currently used in production: Chubby and ZooKeeper. Chubby is a lock service that exposes a file-system-like interface, including operations to acquire and release locks. Chubby is a replicated service and has the master of a replica group initiating all operations to guarantee that they are linearizable. Chubby also enables clients to cache the data of frequently accessed files and manages such caches directly, invalidating cached content explicitly upon updates. ZooKeeper also exposes a file-system-like interface and is a replicated service, but makes different design choices. ZooKeeper is not a lock service and it does not guarantee that all operations are linearizable. Instead it guarantees FIFO order for the operations of individual clients and linearizability of update operations (operations that change the state of ZooKeeper). A weaker consistency guarantee, however, does not imply a reduction of expressiveness when it comes to implementing synchronization primitives. As for client-side caches, ZooKeeper does not manage them directly, and instead uses watches to notify clients of changes.

We finally present ZooKeeper evaluation results and discuss our experience to date with ZooKeeper in production. Our evaluation results show that we can obtain thousands of operations per second with read-dominant workloads, which meets the requirements of current applications.

Orador. Flávio Junqueira is a research scientist with Yahoo! Research in Barcelona. He holds a PhD degree from University of California San Diego in computer science, a MSc degree in Electrical Engineering from COPPE/UFRJ in Rio de Janeiro, and a BS degree cum laude in Electrical Engineering from UFRJ in Rio de Janeiro. His research work has been concentrated on distributed systems and algorithms, and he has worked on projects related to the modeling of failures and vulnerabilities in distributed systems, to the design of distributed algorithms, and to information retrieval in large-scale distributed systems. He is the recipient of a number of awards and

nominations, such as the CSE Department best PhD dissertation award, a nomination to the ACM PhD Dissertation award, and the best paper awards at ACM CIKM 2009 and USENIX ATC 2010. He is an active contributor to open source projects, such as Hadoop and ZooKeeper from the Apache Software Foundation.

Hands on a verification challenge: proving a journaled file system correct

Resumo. In the context of the Verifiable File System (VFS) challenge put forward by Rajeev Joshi and Gerard Holzmann, this talk will address the refinement of an abstract file store model into a journaled (flash) data model catering for wear leveling and recovery from power loss. Such refinement steps are carried out within a simple verification life-cycle where model checking in Alloy goes hand in hand with manual proofs carried out in the (pointfree) algebra of binary relations. This provides ample evidence of the positive impact of Alloy's lemma *everything is a relation* on software verification, in particular in its entailing induction-free proofs about data structures such as stores and lists. (Joint work with Miguel Ferreira, SIG, Amsterdam).

Orador. José Nuno Oliveira (`http://www.di.uminho.pt/~jno`) is currently Associate Professor of Computer Science. He graduated from the U.Porto and received his MSc and PhD in Computer Science from the U.Manchester, where he became interested in formal methods and transformational techniques. He is a member of the Formal Methods Europe (FME) association, where he convenes a subgroup on education, and of the scientific committee of the MAP-i doctoral programme. He is also a member of the IFIP WG2.1 - Algorithmic Languages and Calculi. Since his PhD work on data-flow program transformation, he became interested in program calculi and transformational design. He has served the programme committee of several conferences in the series of FME symposia, MPC, TFM, SEFM, SBMF, ICTAC, ESOP, etc. He was co-chair of MPC'00, FME'00 and TFM'09.

Grammar inference technology applications in software engineering

Resumo. There are many problems whose solutions take the form of patterns that may be expressed using grammars (e.g., speech recognition, text processing, genetic sequencing, programming language development, etc.). Construction of these grammars is usually carried out by computer scientists working with domain experts. Grammar inference (GI) is the process of learning a grammar from examples, either positive (i.e., the pattern should be recognized by the grammar) and/or negative (i.e., the pattern should not be recognized by the grammar).

This talk will present the application of grammar inference to software engineering, including recovery of domain-specific language (DSL) specifications from example DSL programs and recovery of a meta model from instance models which have evolved independently of the original meta model.

Further details are available at `http://www.cis.uab.edu/softcom/GrammarInference`.

This talk was scheduled in the context of the CoRTA Session (Track on *Compilers,Programming Languages, Related Technologies and Applications*) and kindly open to all participants to INForum 2010.

Orador. Barrett Bryant has a B.S. in Computer Science, University of Arkansas at Little Rock (1979); M.S. in Computer Science, Northwestern University (1980); Ph.D. in Computer Science, Northwestern University (1983).

He was Assistant Professor of Computer and Information Sciences, The University of Alabama at Birmingham (1983-1988); 1 Associate Professor of Computer and Information Sciences (Tenured), The University of Alabama at Birmingham (1988-2001); Associate Chair of Computer and Information Sciences, The University of Alabama at Birmingham (since 1996); Professor of Computer and Information Sciences, The University of Alabama at Birmingham (since 2001). He was also a Visiting Researcher, Department of Information Science, Ibaraki University, Hitachi, Japan (1987); Visiting Scientist, Advanced Systems Division, IBM Palo Alto Scientific Center and Data Base Technology Institute, IBM Santa Teresa Laboratory (1991-1992); Research Associate, Air Force Office of Scientific Research, Rome Laboratory, Rome, New York (1993); Visiting Scientist, Army Research Laboratory, Software Technology Branch, Atlanta, Georgia (1995, 1997 and 1998); Visiting Faculty Researcher, Oak Ridge National Laboratory, Oak Ridge, Tennessee (1996); Visiting Faculty Researcher, NASA Marshall Space Flight Center, Huntsville, Alabama (1999); Visiting Associate Professor, Department of Computer Science, Naval Postgraduate School, Monterey, California (2000-2001); Visiting Professor, Department of Computer Science, Asia University, Taichung, Taiwan (2006).

Barrett Bryant major research areas are *Concepts and Implementation of Programming Languages, Formal Language Definition, Grammar-based Systems, Compiler Generators, Domain-Specific Languages, Formal Specification of Software Systems* and *Automated Software Engineering.*

Ciência e Engenharia de Software

Distributed Work Stealing for Constraint Solving
(Extended Abstract)

Vasco Pedro and Salvador Abreu

Departamento de Informática, Universidade de Évora and
CENTRIA FCT/UNL, Portugal
{vp,spa}@di.uevora.pt

Abstract. With the dissemination of affordable parallel and distributed hardware, parallel and distributed constraint solving has lately been the focus of some attention. To effectually apply the power of distributed computational systems, there must be an effective sharing of the work involved in the search for a solution to a Constraint Satisfaction Problem (CSP) between all the participating agents, and it must happen dynamically, since it is hard to predict the effort associated with the exploration of some part of the search space. We describe and provide an experimental assessment of an implementation of a work stealing-based approach to parallel CSP solving in a distributed setting.

1 Introduction

Constraints are used to model problems with no known polynomial algorithm, but for which search techniques developed within the field of constraint programming provide viable procedures. Besides classical applications, such as planning and scheduling, constraints have recently been successfully applied in the contexts of bioinformatics and computer network monitoring [11, 12].

Notwithstanding their relative efficiency, constraint solving methods are computationally demanding and good candidates to benefit from multiprocessing. Moreover, the declarative style of constraint programming frees the programmer from concerns usually entailed by parallel and distributed programming, such as control, synchronisation, and communication issues. In fact, the programmer may not even be aware that there is any parallelism involved in solving the problem. Given the increasing availability of parallel computational resources, in the form of multiprocessors, clusters of computers, or both, there is a need for an effective way to help incorporating that power into the constraint programming setting.

Constraint solving involves exploring large search spaces. To perform search using several agents in parallel, the search effort must be shared among them. In *distributed constraint solving*, in the context of solving Distributed CSPs [17], each agent does a part of the work and coordinates with the other agents in order to find a solution. The present work follows the *parallel constraint solving*

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 7–18

approach [4, 15, 13, 7, 2], where the search space is partitioned and the search for a solution is carried out in each of the sub-search spaces by one agent (or worker), all agents working in parallel. Here the agents are mostly independent from each other, performing their (non-overlapping) part of the work and hoping that one of them will find a quicker path to an answer. While the first approach typically requires significant inter-agent communication, not only for the search to progress but also for termination detection, in the latter communication can be limited to an initial dispatching of the agents and to an answer collecting phase at the end of the procedure. In this case, however, the initial work distribution may turn out to be quite unbalanced, leaving some agents to bear most of the effort as others become idle and their contribution is wasted.

This article reports on preliminary results of our experiments in implementing a work-stealing scheme for overcoming the effect described above. This is a two-level scheme: work stealing occurs between co-located agents, but when distant agents are involved, some cooperation is needed to redistribute the work still left.

The remainder of this paper is structured as follows: we start by establishing some terminology in the next section. Then, in Sections 3 and 4 we describe the architecture of the implemented solver and report on some experimental results obtained with it. Section 5 discusses related work and in Section 6 we conclude and put forward possible continuation paths for this work.

2 Constraint Solving

A constraint satisfaction problem can be briefly defined as a set of variables whose values, to be drawn from their domains, must satisfy a set of relations.

Definition 1 (CSP). *A* Constraint Satisfaction Problem *(CSP) over finite domains is a triple $P = (X, D, C)$, where*

- $X = \{x_1, x_2, \ldots, x_n\}$ *is an indexed set of* variables*;*
- $D = \{D_1, D_2, \ldots, D_n\}$ *is an indexed set of finite sets of values, with D_i being the* domain *of variable x_i, for every $i = 1, 2, \ldots, n$; and*
- $C = \{c_1, c_2, \ldots, c_m\}$ *is a set of relations between variables, called the* constraints*.*

The *search space* of a CSP consists of all the tuples from the cross product of the domains, where each variable is assigned a value from its domain. Solving a CSP amounts to finding some or all of those tuples which satisfy all constraints of the problem.

Definition 2 (Solution). *A* solution *to a CSP is an n-tuple $(v_1, v_2, \ldots, v_n) \in D_1 \times D_2 \times \ldots \times D_n$ such that all constraints are satisfied.*

In parallel constraint solving, the problem is divided into subproblems. Solutions to these subproblems are also solutions to the original problem.

Definition 3 (Subproblem). *A* subproblem *of a CSP* $P = (X, D, C)$ *is a CSP* $P' = (X, D', C)$ *such that* $D' = \{D'_1, D'_2, \ldots, D'_n\}$ *and* $D'_i \subseteq D_i$*, for every* $i = 1, 2, \ldots, n$*.*

To guarantee completeness of the search, the search spaces of the subproblems must cover the search space of the original problem. In order to avoid redundant work, they must also be pairwise disjoint.

Definition 4 (Partition). *A set* $\{P'_1, P'_2, \ldots, P'_k\}$ *of subproblems of a CSP* P*, with* $P'_i = (X, \{D'_{i1}, D'_{i2}, \ldots, D'_{in}\}, C)$*, is a* partition *of* P *if*

$$\bigcup_{1 \leq i \leq k} D'_{i1} \times D'_{i2} \times \cdots \times D'_{in} = D_1 \times D_2 \times \cdots \times D_n$$

and $(\forall\, i \neq j)\, D'_{i1} \times D'_{i2} \times \cdots \times D'_{in} \cap D'_{j1} \times D'_{j2} \times \cdots \times D'_{jn} = \emptyset$*.*

A partition of a CSP may be dually regarded as a partition of its search space, the search spaces of the subproblems being sub-search spaces of the original problem. In this paper we will only deal with search space partitions that correspond to some partition of a problem.

3 Solver Architecture

Our constraint solver consists of *workers*, grouped together as *teams* (Figure 1). The search for one or all solutions is carried out by the workers, which implement a propagator based constraint solving engine, following a domain consistency oriented approach [1]. Each active worker has a *pool* of *idle* search spaces and a *current* search space, the one it is currently exploring. In each team there is a *controller*, which does not participate in the search, and one of the controllers, the *main controller*, also coordinates the teams.

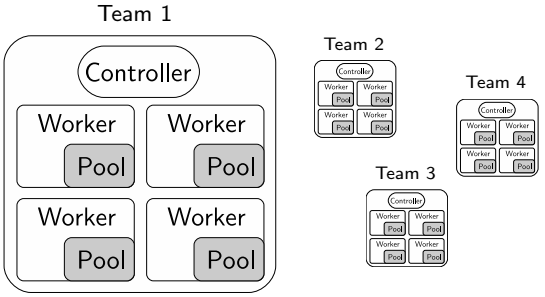


Fig. 1. Solver architecture

Structuring the workers this way serves two purposes: the first is that a workers' sole task becomes searching, as all communication with the environment required by the dynamic sharing of work among teams is handled by the

controller. The second objective is the sharing of resources enabled by binding the workers in a team close together. If all workers were on the same level, they would either have to divide their attention between search and communication or there would have to be one controller per worker, thereby increasing resource usage. On the other hand, this structure matches naturally a two-level partitioning of the search space and we obtain receiver-initiated decentralised dynamic load balancing [16].

At the outset of the search process, the problem to be solved is partitioned and each team is entrusted with trying to solve one of the resulting subproblems. The controller in each team then partitions the local problem and hands each sub-search space over to a worker for exploration.

On finishing exploring its assigned search space, a worker tries to steal work from another worker *within its team*. If unsuccessful, it then notifies the team controller that it has become idle. When all the workers in a team are idle, the controller asks the other teams for more work.

3.1 Partitioning Strategies

The strategy used to partition the search space has a decisive impact on the number of steps needed to get to a solution, hence on performance.

Partitioning strategies may be designed either to lead to a balanced distribution of the search work, like the *even* strategy below and the prime and greedy strategies from [14], or to produce some subproblems where the search is expected to be quick (while others may be slow), such as *eager* partitioning. In principle, the former strategies will be more suited to situations where all solutions are requested and the whole search space must be visited, and the latter will lend themselves better to when one solution is enough. In any case, the splitting of the problem will introduce a breadth-first component into the usual depth-first exploration of the search tree, which sometimes gives rise to superlinear speedups.

In *even partitioning*, domains are split so as to obtain sub-search spaces of similar dimensions. If we want to split a problem into k subproblems, then the first variable with at least that many values in its domain is chosen and its domain is split as evenly as possible among the subproblems: if the domain of the chosen variable has $d \geq k$ values, then it will have $\lfloor d/k \rfloor$ values in the first $k - d \bmod k$ subproblems and $\lfloor d/k \rfloor + 1$ values in the remaining $d \bmod k$ subproblems.

Eager partitioning corresponds roughly to a partial breadth-first expansion of the search tree and it will mostly produce subproblems where at least one of the variables has had its domain reduced to a single value. The splitting is performed according to the algorithm depicted in Figure 2, whose inputs are the number of subproblems to create and a sequence of problems from which to create them. Initially, this sequence only contains the original problem.

The partitioning of the CSP may affect the behaviour of the search, even to the point of defeating the variable and value selection heuristics which are usually appropriate to a given problem, as has been noted in [7, Section 6]. This

Notation *If P is a CSP and D is a finite set, PD_i stands for the CSP which is identical to P except that the domain of the i^{th} variable is D.*

```
eager-split(k, (P_1 P_2 ··· P_q))
  (X, D, C) ← P_1
  i ← min {j | |D_j| > 1}
  d ← |D_i|
  {v_1, v_2, ..., v_d} ← D_i
  if k ≤ d then
    (P_1{v_1}_i P_1{v_2}_i ··· P_1{v_k, ..., v_d}_i P_2 ··· P_q)
  else
    eager-split(k − d + 1, (P_2 ··· P_q P_1{v_1}_i P_1{v_2}_i ··· P_1{v_d}_i))
```

Fig. 2. Eager partitioning algorithm

suggests that the partitioning strategy, introducing another degree of freedom in the search strategy, needs to be adapted to the problem being solved and matched with the search heuristics used, and that no overall 'best' partitioning strategy exists. (Notice that, for the present, problem specific heuristics do not inform problem partitioning.)

As problem partitioning takes place at two points in the process — to distribute work to all the teams, and, initially within every team, to assign work to each worker — different splitting strategies can be used, a more balanced one to allot similar amounts of work to the individual teams, and another to focus the efforts of the agents. The latter strategy could be finer grained than the former, the cost of local work stealing being much lower than that of network supported work sharing.

3.2 Search

The search unfolds as a worker further splits the search space it is working on, keeping one part as its current search space and adding the other to its pool of idle search spaces. If the current search space is found to contain no solution, the worker draws a new search space from the pool and starts exploring it, never backtracking. Upon finding a solution, the worker communicates it to the team controller which, in turn, forwards it to the main controller.

The state of a worker with two search spaces currently in the pool is shown in Figure 3, where solid edges mean that the child search spaces form a partition of the parent. Notice that the subtree to the left of the current search space (corresponding to the tuples where both x_1 and x_2 take value 1) has already been explored and discarded, and is not displayed.

Figure 4 depicts the main driver algorithm for workers. At each step of the search process, a worker starts by looking within its current search space for a variable whose domain is not a singleton (line 3). If none is found, then the search space contains a single tuple which constitutes a solution to the problem,

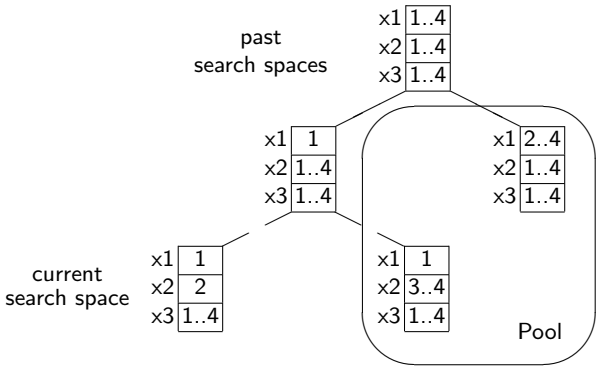


Fig. 3. Search spaces from a worker

and which is returned by the worker (line 10). Otherwise, one of the variables with a non-singleton domain is selected and the current search space is split into two subspaces (line 4):

- In the first, which will become the worker's current search space, the selected variable is set to an individual value picked from its domain.
- In the other, to be added to the pool of idle search spaces (line 5), that value is removed from the domain of the variable.

The domains of the other variables remain unchanged in both search spaces.

Following the split, the new current search space goes through a propagation phase (line 6). If it succeeds, another search step is performed. If the propagation fails, the worker tries to fetch an idle search space from the pool to become the current search space (line 7). If this is not possible the worker fails (line 9), otherwise the search resumes with the retrieved search space undergoing a propagation phase, as the domain of one of its variables shrunk just prior to it being stored in the idle pool.

```
 1: WORKER(search-space)
 2:    current ← search-space
 3:    while var ← select-variable(current) do
 4:       (current, other) ← split-search-space(var, current)
 5:       pool-put(other, var)
 6:       while (current ← revise(var, current)) = FAIL do
 7:          (current, var) ← pool-get()
 8:          if current = FAIL then
 9:             return FAIL
10:    return SOLUTION(current)
```

Fig. 4. Worker main driver algorithm

3.3 Work Stealing

When a worker tries to fetch a new search space from its pool and finds it empty, it will attempt to obtain one from one of its teammates. In order to minimise the impact on the performance of the solver, this is achieved with as little cooperation from the holder of the retrieved search space as possible. In fact, the idle worker will effectively steal work from a teammate while the latter continues its task, oblivious to what is being done to its work queue.

The intended discipline of a worker's pool is that of a *deque* (double-ended queue), as depicted in Figure 5. While the owner works on one end of its pool (lines 2, 8, and 12), a worker whose pool is empty will remove an entry from the other end (line 20). This way, the only penalty a worker incurs during normal processing is the cost of an extra check on the size of its pool (line 6). The protocol used to avoid interference during pool accesses is similar to the one in [5]. Only when the number of entries in the pool is small, will it be necessary to enforce mutual exclusion in the accesses to the pool, and even then only when removing a search space. To reduce contention, work stealing is only allowed from a pool when the number of entries in it reaches a given threshold (line 17).

```
 1: pool-put(search-space, variable)
 2:     pool.append(search-space, variable)

 3: pool-get()
 4:     if pool.size = 0 then
 5:         return steal-work()
 6:     else if pool.size < SAFE-SIZE then
 7:         lock(pool)
 8:         ss ← pool.remove-last()
 9:         unlock(pool)
10:         return ss
11:     else
12:         return pool.remove-last()

13: steal-work()
14:     lock(stealing)
15:     v ← worker-with-biggest-pool()
16:     lock(v.pool)
17:     if v.pool.size < THRESHOLD then
18:         ss ← FAIL
19:     else
20:         ss ← v.pool.remove-first()
21:     unlock(v.pool)
22:     unlock(stealing)
23:     return ss
```

Fig. 5. Pool insertion and removal, and work stealing algorithm

Stolen work corresponds to locations nearer the root of a worker's search tree. The search within the worker's search space proceeds according to the heuristics deemed adequate to the problem until it either finds a solution or the work is exhausted. Upon stealing work from a peer, a worker picks up the search at a point that the worker it was stolen from would eventually reach, thus subverting the problem's search strategy and introducing in it a measure of randomness. This may be either beneficial or detrimental, depending on the specific problem.

In the event of an idle worker failing to obtain work within its team, it notifies the team controller and waits, either to be later restarted or to be terminated. When all the agents in a team have become idle, the team controller broadcasts a request for more work to the other teams.

Inter-team work stealing follows along a simple plan: initially, one of the team controllers is given the role of fulfilling requests for work. Upon receiving one, and using the same protocol used by the workers, it tries to steal a search space from the local pool to be forwarded to the requester, which splits it among its workers and becomes the new work supplier. If the designated work supplier is unable to spare a search space, the remaining teams are polled for work, as done in [13]. When no team is able to supply additional work, the idle team notifies the main controller and terminates.

3.4 Implementation Notes

One of the main goals behind this work was to build a constraint solver which could take advantage of the advances in parallel architectures and in clustering network technology. To better be able to handle the challenges inherent to multiprocessing, namely memory management and caching issues, C was our choice for the implementation language, as it allows for very fine-grained control.

Teams are autonomous entities and each team corresponds to a distinct process, usually residing on a dedicated machine. As communication, particularly over a network, may have an adverse impact on system performance, care has been taken to minimise the number of inter-team messages needed. Teams are coordinated by way of an IPC library.

A team comprises active components which are the workers and the controller. The controller is, most of the time, waiting for a worker or another team controller to communicate with it, not disturbing the search process and allowing workers to be mapped to processors. Workers are mostly independent from each other, except where work stealing is concerned, as explained in Section 3.3. A worker, to be able to steal work from another one without active cooperation from the latter, must be able to access all the team pools. To make this possible, pools are located in shared memory and workers, as well as the controller, are implemented as lightweight processes (threads).

4 Experimental Results

In this Section, we present some performance results obtained with our solver on two classic benchmark problems, namely the non attacking queens problem and the Langford number problem [3, problem 024]. While the queens problem has many solutions well spread out throughout the search space, the Langford number problem either has no solution or it also has many solutions but not so well distributed.

Measurements were made of the time taken to count all solutions for the two problems and for generating the first solution in the second problem. These measurements were made on a cluster of Q6600 Intel Core2 Quad CPUs, clocked at 2.4GHz, with 2–4GB RAM, running Linux, and the code was compiled with GCC 4.1.1 with the '-O3' flag. The times presented are the average of 12 runs of each program, with the worst and the best times excluded. When computing

the relative performance with respect to the sequential case, we subtracted the overhead associated with starting up and terminating the solver, which reached a maximum of 0.3 seconds in the 6 teams configuration. Unless otherwise indicated, teams are composed of 4 workers, mirroring the number of CPUs in the shared-memory multiprocessor systems. For interprocess communication, the Open MPI MPI-2 implementation [9] was used.

Absolute performance has not, so far, been the top priority goal of this work. Nevertheless the sequential (1 team with 1 worker) version of our solver already displays interesting times for solving these problems, as attested by Table 1, where they are compared with those of Gecode 3.0.2 [6], although there clearly remains some work to be done in that regard.

Table 1. Times comparison with Gecode (seconds)

	Queens			Langford				
	14	15	16	2 11	2 12	2 28	2 31	3 18
Our solver	13.89	86.05	580.36	1.07	8.00	67.56	1.26	2.44
Gecode	17.21	102.18	646.43	36.40	25.01	0.03	0.02	0.42
	all solutions					first solution		

In the remainder of this section, we look at the results obtained with several configurations of the solver and analyse them with respect to the speedups induced by the parallelisation of the search, using the two partitioning strategies. The use of the two strategies helps both to illustrate the consequences of problem partitioning and to highlight the effect of work stealing.

In the non attacking queens problem, the first observation that can be made in relation to the speedups obtained, depicted in Figure 6, is that they are fairly insensitive to the partitioning strategy used. Given that in this problem the work is very evenly distributed among the possible values from the domains of the variables, this result is only possible due to effective work sharing.

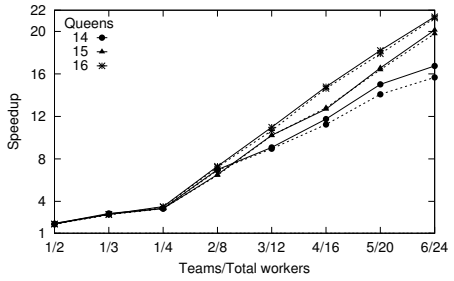
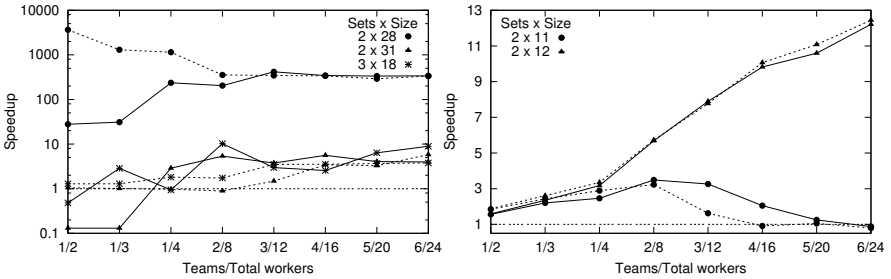


Fig. 6. Speedups for the non attacking queens (all solutions)[1]

[1] In these graphs, solid and dashed lines correspond, respectively, to even and eager partitioning.

The profile of the speedups evolution with the addition of more teams is quasi-linear for the 16 queens problem, showing good scalability of the approach. However, the smaller problem starts suffering from the weight of the implementation early on. Total running times for the three problems in the 6 team setting are around 1.2, 4.6, and 27.5 seconds, for 14, 15, and 16 queens, respectively.

The Langford number problem, for which we measured both the speedups for counting all solutions and for obtaining the first solution, is an example of a case where domain partitioning interacts badly with the heuristics usually used for guiding the search, as dividing a domain gives rise to more work than that needed to solve the original problem. This is apparent in Figure 7a, which represents the results observed in finding the first solution and where some instances of the problem displayed a marked slowdown when partitioning the domain of the first variable in two or three similarly sized parts. On the other hand, speedups of more than 3000 were also obtained in one case.



(a) First solution **(b)** All solutions

Fig. 7. Speedups for the Langford number problem

Counting all solutions of the Langford problem (Figure 7b) exhibits a profile common to the previous problem, but at some point the implementation starts overwhelming the potential improvements due to the parallelisation on the smaller instance. This effect requires further study to identify and solve its causes.

5 Related Work

Recent years have seen an increase in the interest in parallel solving, as parallel architectures become more common. An early language sporting parallel constraint solving was the CHIP parallel constraint logic programming language [15]. It was implemented on top of the logic programming system PEPSys, whose or-parallel resolution infrastructure was adapted to handle the domain operations needed in parallel constraint solving.

More recent works rely on features of an underlying framework for programming parallel search. The concurrent Oz language provides the basis for the

implementation described in [13], where search is encapsulated into computation spaces and a distributed implementation allows the distribution of workers. Work sharing is coordinated by a manager, which receives requests for work from the workers and then tries to find one willing to share the work it has left. Search strategies are user programmed and the work sharing strategy is implemented by the workers.

A similar approach is taken in [7, 8] which show how to program parallel search controllers in COMET. There, the pool is an active object which is queried by the idle workers. In case the pool is empty, it asks another worker to generate yet unexplored sub-search spaces, gives one away and stores the rest. It is not explained, however, how the worker which supplies work is chosen.

A focus of research has been on the strategies for splitting the work between workers. These strategies may be driven by the problem structure, such as the size of the domains [14], or by the past behaviour of the solver, be it related with properties of the solving process, such as the number of variables already instantiated [10], or with the progress of the search, in what it affects the prospects of finding a solution in the current subtree [18] or in the subtrees left to explore [2]. Here, a scheme is presented which uses the search heuristics to guide problem splitting, dampened by a degree of confidence to distribute the workers across the search tree while maintaining some bias towards the nodes favoured by the heuristic. It shows good performance on multi-core hardware, and while it has the drawback of working on a global view of the search process, it seems to point in a promising direction of research, namely using the work done as a guide to future search space splitting.

6 Conclusions and Future Work

In spite of the results obtained so far, there should be additional gains with a more sophisticated work sharing protocol. Several possibilities should be studied, including having a different work stealing policy for inter-team sharing, where candidate search spaces undergo a deeper examination to try to determine whether the cost of their sending is offset by the work saved locally.

Short term development plans comprise improving the internal representation of the domains, which currently only allows values between 0 and 63, the inclusion of optimisation constraints, and the improvement of the scalability of the implementation in two key aspects: the initial work distribution and the sharing of work between teams, which could both profit from organising the teams in some way.

We also plan on experimenting with different underlying models and libraries for thread management and inter-process communication, namely to venture beyond the present implementation which relies on Posix threads and MPI.

Acknowledgements

The authors wish to acknowledge the FCT/Pessoa grant 'CONTEMP — CONTraintes Exécutées en MultiProcesseurs' and the members of the partner IN-

RIA/Bordeaux RUNTIME team, namely Olivier Aumage, Jérôme Clet-Ortega, and Cédric Augonnet, for their cooperation and helpful suggestions. Thanks are due to Miguel Avillez at Universidade de Évora for the valued offer of computational support. The authors would also like to thank the anonymous reviewers for their comments and suggestions.

References

1. Bessière, C.: Constraint propagation. In: Rossi et al. [11], chap. 3, pp. 29–83
2. Chu, G., Schulte, C., Stuckey, P.J.: Confidence-based work stealing in parallel constraint programming. In: Gent, I.P. (ed.) CP'09. LNCS, vol. 5732, pp. 226–241. Springer, Lisboa, Portugal (Sep 2009)
3. CSPLib: A problem library for constraints. `http://www.csplib.org/`.
4. Ferreira, L.: Programação por Restrições Distribuídas em Java. Ph.D. thesis, Universidade de Évora, Portugal (2004)
5. Frigo, M., Leiserson, C.E., Randall, K.H.: The implementation of the Cilk-5 multithreaded language. In: PLDI'98. pp. 212–223. ACM, Montreal, Quebec, Canada (Jun 1998)
6. Gecode: Generic constraint development environment. `http://www.gecode.org/`.
7. Michel, L., See, A., Van Hentenryck, P.: Parallelizing constraint programs transparently. In: Bessière, C. (ed.) CP'07. LNCS, vol. 4741, pp. 514–528. Springer, Providence, RI, USA (Sep 2007)
8. Michel, L., See, A., Van Hentenryck, P.: Transparent parallelization of constraint programming. INFORMS Journal on Computing 21(3), 363–382 (Dec 2008)
9. Open MPI Project: `http://www.open-mpi.org/`.
10. Rolf, C.C., Kuchcinski, K.: Load-balancing methods for parallel and distributed constraint solving. In: 2008 IEEE Int. Conf. on Cluster Computing. pp. 304–309 (2008)
11. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Foundations of Artificial Intelligence, Elsevier (2006)
12. Salgueiro, P., Abreu, S.: Network monitoring with constraint programming: Preliminary specification and analysis. In: Abreu, S., Seipel, D. (eds.) INAP2009. pp. 37–52. Évora, Portugal (Nov 2009)
13. Schulte, C.: Parallel search made simple. In: Beldiceanu, N., Harvey, W., Henz, M., Laburthe, F., Monfroy, E., Müller, T., Perron, L., Schulte, C. (eds.) TRICS-2000. pp. 41–57. Singapore (Sep 2000)
14. Silaghi, M.C., Faltings, B.: Parallel proposals in asynchronous search. Tech. Rep. TR-01/371, Swiss Federal Institute of Technology (EPFL), Lausanne (Aug 2001)
15. Van Hentenryck, P.: Parallel constraint satisfaction in logic programming: Preliminary results of CHIP within PEPSys. In: Levi, G., Martelli, M. (eds.) ICLP'89. pp. 165–180. The MIT Press, Lisboa, Portugal (Jun 1989)
16. Wilkinson, B., Allen, M.: Parallel Programming. Pearson, 2nd edn. (2005)
17. Yokoo, M., Durfee, E.H., Ishida, T., Kuwabara, K.: The distributed constraint satisfaction problem: Formalization and algorithms. Trans. on Knowl. and Data Eng. 10(5), 673–685 (1998)
18. Zivan, R., Meisels, A.: Concurrent search for distributed CSPs. Artificial Intelligence 170(4–5), 440–461 (Apr 2006)

JFly: A JML-Based Strategy for Incorporating Formal Specifications into the Software Development Process

Nestor Catano[1], João Pestana[2], and Ricardo Rodrigues[2]

[1] The University of Madeira, M-ITI, CMU-Portugal
ncatano@m-iti.org
[2] The University of Madeira, Portugal
{joao.pestana,ricardo.rodrigues}@max.uma.pt

Abstract. This paper presents JFly, a JML-based strategy for incorporating formal specifications into the software development of object oriented programs. The strategy consists in evolving functional requirements into a semi-formal requirements form, and then expressing these requirements as JML formal specifications. What makes our strategy different from existing strategies is the particular use of JML we make all along the way from requirements to validation-and-verification. We validate our strategy with the formal development of a smart card application for managing medical information.

1 Introduction

Although software engineering methods provide a disciplined approach to software development, it is still quite common to find flawed software systems. A way to construct correct programs is through the use of formal specifications as part of a software engineering practice. In this paper, we propose JFly, a strategy that incorporates formal specifications into the software development process of object oriented programs by evolving informal functional requirements into formal specifications (Section 2). Having informal functional requirements modelled in a formal specification language allows for the use of formal methods tools for checking program correctness. We use JML [11] as the formal specification language for writing our specifications. Our strategy consists in evolving informal functional software requirements written in natural language (e.g. English or Portuguese) into semi-formal requirements, i.e. requirements written in natural language, yet in a more mathematical style. Hence, informal requirements are written as semi-formal functional requirements of the form if <event> then <restriction>, or as semi-formal class and system invariants. Semi-formal requirements are then written as JML method specifications and JML class invariants respectively. We validate our strategy with the development of a HealthCard smart card application (Section 3). We used the JML Common tools [3] to check the HealthCard for correctness. Neither using formal specifications to increase confidence in a system implementation nor gradually transforming software requirements from a high level of abstraction to a more concrete level are new

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 19–30

ideas [6, 8, 10]. What makes our strategy different from others is the particular use of JML specifications we make all along the way from requirements to validation-and-verification.

L. Shaoying et al. in [15] propose the SOFL methodology for software development. The SOFL language integrates Data Flow Diagrams, Petri Nets, and VDM-SL. Likewise JFly, in SOFL, the process of writing formal specifications is the result of a 3-step process that evolves informal specifications into an abstract formal specification. The SOFL methodology has established a much more mature technique for developing formal design specifications for software systems than the JFly strategy presented by us. Nonetheless, having formal specifications expressed directly in JML, rather than using an abstract general form, allows the direct use of JML tools, which implement various formal checking techniques. JML is a formal specification language that uses a syntax close to Java syntax and hence Java programmers find JML easy to use, which helps to bridge the gap between mathematical formalisms and software engineering techniques [4].

Supplementary to our work, V. T. Vasconcelos et al. have implemented the ConGu tool [16], which reduces the problem of checking algebraic specifications to the run-time monitoring of contracts described in JML. Their work extend our work in a way that further considers algebraic specifications to express correct software components.

Finally, M. G. Ilieva and O. Ormandjieva have studied the automatic translation of software requirements written in natural language into formal specifications [9]. Our work is less ambitious, yet more practical.

1.1 The Java Modeling Language (JML)

JML is a behavioral interface specification language for Java, which means that the only correct implementation of a JML class specification is a Java class implementation with the specified behavior. JML is now an academic community effort with many groups developing tools to support JML [3]. In JML, methods are specified using `requires`, `modifies`, and `ensures` clauses, which give the precondition, the frame (what locations may change from the pre- to the poststate), and the postcondition respectively. A method specification can also include an `exsures` or `signals` clause to specify conditions under which the method could throw an exception. Class invariants can also be written to constrain the states of objects. JML specifications use Java syntax and are embedded in Java code between special comments `/*@ ... @*/` or after `//@`. A simple JML specification for a Java class consists of pre- and post-conditions added to its methods, and class invariants restricting the possible states of class instances. Specifications for method pre- and post-conditions are embedded as comments immediately before method declarations. JML predicates are first-order logic predicates formed of side-effect free Java boolean expressions and several specification-only JML constructs. Because of this side-effect restriction, Java operators like `++` and `--` are not allowed in JML specifications. JML provides notations for forward and backward logical implications, `==>` and `<==`, for non-equivalence `<=!=>`, and for

logical *or* and logical *and*, `||` and `&&`. The JML notations for the standard universal and existential quantifiers are `(\forall T x; E)` and `(\exists T x; E)`, where `T x;` declares a variable `x` of type `T`, and `E` is the expression that must hold for every (some) value of type `T`. The expressions `(\forall T x; P; Q)` and `(\exists T x; P; Q)` are equivalent to `(\forall T x; P ==> Q)` and `(\exists T x; P && Q)` respectively.

1.2 The JML Common Tools

The JML common tools [3, 2] is a suite of tools providing support to run-time assertion checking of JML-specified Java programs. The suite includes `jml`, `jmlc`, `jmlunit` and `jmlrac`. The `jml` tool checks the JML specifications for syntax errors. The `jmlc` tool compiles JML-specified Java programs into a Java bytecode that includes instructions for checking JML specifications at run-time. The `jmlunit` tool generates JUnit [12] unit tests code from JML specifications and uses JML specifications processed by `jmlc` to determine whether the code being tested is correct or not. Test drivers are run by using the `jmlrac` tool, a modified version of the `java` command that refers to appropriate run-time assertion checking libraries.

The JML common tools make possible the automation of regression testing from the precise, and correct JML characterisation of a software system. The quality and the coverage of the testing carried out by JML depend on the quality of the JML specifications. The run-time assertion checking with JML is sound, *i.e.*, no false reports are generated. The checking is however incomplete cause users can write informal descriptions in JML specifications, e.g., `(* x is positive *)`. The completeness of the checking performed by JML depends on the quality of the specifications and the test data provided.

2 JFly: the Proposed Strategy

We have developed a strategy for evolving informal functional requirements into formal specifications, which can be employed as part of existing object-oriented software development methodologies [14] (Chapter 28). Hence, software developers must define precise interface specifications for underlying software components, based on data types and the conceptual metaphor of the design-by-contract [13]. The strategy consists in incorporating informal, semi-formal, and formal specifications all along an existing object-oriented software engineering methodology. In Figure 1, we do not restrict any phase to occur before or after any other phase, so that arrows convey information on usage rather than on precedence in time. Software development phases are iterative so that they can be revisited at later phases to obtain a correct implementation of the system. During the analysis phase, "informal" functional requirements are written (functional requirements written in natural language). As informal functional requirements are expressed in a natural language, inconsistencies can be introduced during the analysis phase. Hence, informal functional requirements are

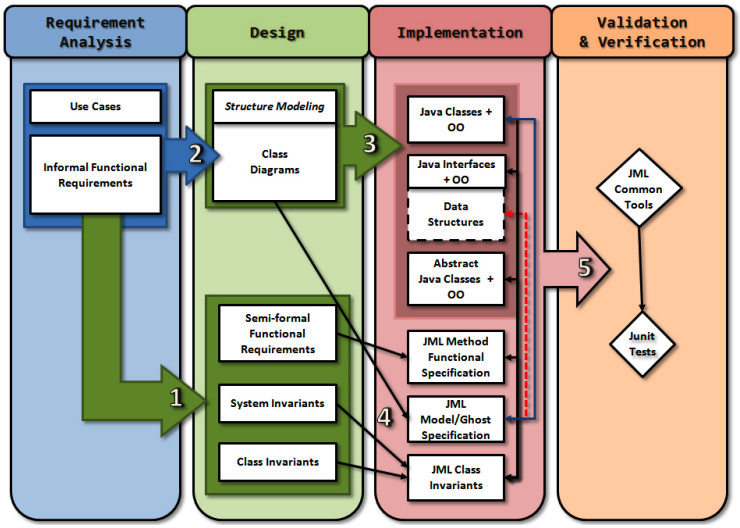


Fig. 1. The Software Development Process

first evolved into "semi-formal" requirements (see Arrow 1), and then ported into JML formal specifications (see Arrow 4). Having formal specifications expressed in JML makes it possible to use JML-based formal methods tools to check for flaws. The semi-formal requirements are divided into three parts, namely the semi-formal functional requirements (ported into JML method specifications), the class invariants, and the system invariants (these two are ported into JML class invariant specifications). Evolving the informal functional requirements into semi-formal ones involves expressing informal requirements into an if <event> then <restriction> form (see Section 3.3 for details).

During the design phase, the requirements gathered from the analysis phase are used to define the structure of the system (see Arrow 2), which is later used to write classes, their attributes, their methods, and the relations among them (see Arrow 3). These classes are specified with JML as described above. During the implementation, we start by writing Java interfaces and Java abstract classes. From the semi-formal requirements, JML functional specifications are written for abstract methods in Java interfaces and abstract classes, and JML class invariants are written, modelling global properties of the system. Finally, JML abstract variables are defined to describe the distinct abstract data types used in the application, and how they are manipulated through class inheritance (see Section 3.5). JML specifications provide support to the writing of correct code for concrete classes that implement the interfaces and the abstract Java classes. JML specifications also provide support to a business contract programming style of programming, in accordance with Bertrand Meyer's design-by-contract principles.

During the validation-and-verification phase, the implementation is checked against the JML specifications (Arrow 5), using the JML Common Tools [3]. If they issue an error, then the implementation or the specifications are evolved accordingly. Therefore, it is possible to go back to a previous phase and make amendments to the JML specifications or the implementation itself. Notice that inconsistencies can be detected before an implementation for the system is written. For instance Java interfaces and Java abstract classes are checked against JML specifications before writing an implementation for these classes and interfaces, or JML specifications can be validated in isolation [5].

3 A Running Example

3.1 The HealthCard Application

In the following, we describe the HealthCard smart card application we used to validate our strategy. HealthCard stores people's medical information. Smart cards are pocket-sized plastic cards with embedded integrated circuits that process data. A typical smart card application includes on-card applets (the applets running on the card), a card reader-side, and off-card applications (e.g. a computer program communicating with the card applets). HealthCard is written in Java Card, a subset of Java used to program card applets. We used the Java Card Remote Method Invocation (JCRMI) model for communication between off-card applications and on-card applets. This model implements a client-server setting with the HealthCard acting as server, and off-card applications as clients, communicating via APDU (Application Protocol Data Unit) messages. Figure 2 shows the structure of the HealthCard. A patient can use his HealthCard to furnish accurate medical information to general practitioners in medical centres with the appropriate system to read it. The HealthCard manages the patient's personal details, his allergies, his historical record of vaccines, diagnosis, treatments and prescribed medicines. The HealthCard is divided in several modules for managing medical information. Each module has a remote interface and an implementation class that serves the appropriate services. All the remote interfaces are referenced in a single remote interface named `CardServices` whereby an external client can invoke services. Hence, if an external client calls the method `getApp()` in `CardServices`, it gets a reference to the `Appointments` remote interface. This reference is then used to invoke appropriate methods implementing services.

3.2 Informal Functional Requirements

Informal functional requirements define, in an informal way, the inputs, the behavior, and, in general, functional restrictions of the system to develop. In the following, we present a small example from the HealthCard system that shows how informal functional requirements are evolved into the three kind of semi-formal requirements described in Section 2. We present below some of the informal requirements of the HealthCard application.

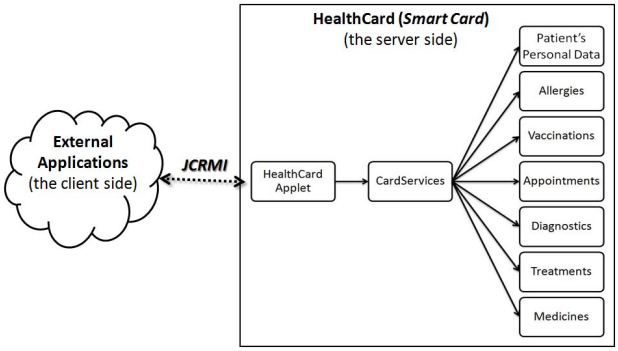


Fig. 2. The Health Card System Structure

IFR1 There must not exist duplicated entries for allergies with the same designation code.
IFR2 A fixed number of allergies can be introduced in the card only.
IFR3 All allergy designation codes must have a stipulated length.
IFR4 The prescription date of a medicine must be bigger than or equal to the date of the appointment in which the medicine was prescribed.

The following sub-sections show how the informal requirements above are evolved into semi-formal requirements. Evolving informal requirements into semi-formal requirements is not a deterministic process, nonetheless it obeys general guidelines that proved to be practical in the development of the HealthCard. Informal requirements involving several domain concepts (e.g. medicine and appointment in IFR4) are evolved into system invariants; informal requirements involving or restricting a single domain concept (e.g. allergy in IFR3) are evolved into class invariants; informal requirements functionally restricting a single domain concepts under certain events (e.g. IFR1 and IFR2) are evolved into semi-functional requirements that follow an if <event> then <restriction> mathematical form, which is close to a JML method specification style.

Because of the ambiguous essence of natural language, our guidelines to transform informal requirements into semi-formal ones is neither sound nor complete, nonetheless our experience shows that is useful in practice. We have implemented a prototype script shell based tool doing the transformation automatically (see Section 3.8).

3.3 Semi-Formal Functional Requirements

The informal functional requirement IFR1 is transformed into if <a new allergy is to be added to the list of referenced allergies and the allergy designation has already been referenced>, then <the new allergy is not inserted>. We show below the semi-formal requirements obtained from the first two informal requirements above.

SFR1 From IFR1. If a new allergy is to be added to the list of referenced allergies and the allergy designation has already been referenced, then the new allergy is not inserted.

SFR2 From IFR2. If an allergy is to be added to the list of referenced allergies and the limit of the number of referenced allergies has already been attained, then the state of the card remains unchanged.

3.4 Class and System Invariants

Class invariants are written from informal requirements that describe limitations or constraints on a small-scale, e.g. limitations of properties that eventually will restrict or describe a certain domain concept only. Hence, the informal functional requirement IFR3: "All allergy designation codes must have a stipulated length", restricts the length of the designation code of any "allergy". To write this class invariant (see CI1 below), we use a variable des to represent the designation code of the allergy. This variable can be modelled as a JML `abstract` variable (see Section 3.5). CI1 describes a property about the length of the code of an allergy, so that eventually it will become a class and the code a `static` field of it.

CI1 invariant size(des) == CODE_LENGTH

Unlike class invariants, system invariants describe invariant properties relating several domain concepts. For instance, IFR4 describes a property on medicines, managing information on prescribed medicines in appointments, and appointments, managing information about appointments scheduling. IFR4 becomes the semi-formal system invariant requirement SI1 below.

SI1 For all object m of type medicine, and all object a of type appointment such that appointment(m) is equals to a, then date(m) is bigger than or equal to date(a).

3.5 Design and Implementation

During the design phase, the structure of the application is created from the requirements. This structure encompasses class diagrams for interfaces, abstract classes, and concrete classes. In parallel to this phase, semi-formal functional requirements and class and system invariants are written (Sections 3.3 and 3.4). Semi-formal specifications are later ported to JML specifications (Section 3.6). During the implementation phase, from the structure of the application generated in the design phase, Java abstract classes, Java interfaces, and Java classes are written. In a first stage, the implementation only contains code skeletons, so no method in any concrete class is implemented. JML specifications are embedded within the code. Hence, the JML Common Tools can be used to check the code during early stages of the implementation (i.e. before fully implementing concrete Java classes). Therefore, the Java code can be evolved so as to conform to the JML specifications, or the specifications can be evolved so as

to conform to an expected behavior. Checking that one conforms to another is done automatically with the JML Common Tools. JML eliminates programmers' responsibility of keeping track of how properties a program must respect are affected by changes in the code.

Furthermore, to have a high level of abstraction in specifications, JML provides support to the use of abstract variables, which exist at the level of the specification, but not in the implementation. Declarations of abstract variables are preceded by the JML keyword `model` and are related to Java code by a `represents` clause[3]. This clause specifies how the value of an abstract variable is calculated from the values of concrete variables (see Section 3.6). Abstract variables are useful in describing properties about interfaces because these are not allowed to declare (concrete) variables in Java. Within an interface, an abstract variable describes the state of the implementing classes. Abstract variable specifications for interfaces and for abstract classes do not need to be written down again in implementing classes or sub-classes, since JML specifications are inherited. This ensures behavioral sub-typing through which a sub-class object can always be used where a super-class object is expected.

3.6 JML Formal Specification

Semi-formal functional requirements SFR1 and SFR2 relate to method `addAllergy` in interface `Allergies` (see below). In Java, interfaces cannot declare attributes, hence `Allergies` declares an `abstract` JML variable `as`, modelling stored referenced allergies. The JML `JMLEqualsSequence` type models a sequence of objects that can be compared using the standard method `equals`. We declare two additional abstract variables, `size` and `maxsize`, modelling the number of referenced allergies and the maximum number of referenced allergies. A normal behavior specification expresses that if all the pre-conditions hold (clauses `requires`) in the pre-state of the method, it will terminate in a state in which all the post-conditions (clauses `ensures`) hold. SFR2 is expressed as the JML pre-condition `size < maxsize`. SFR1 appears in two separated normal postconditions that make use of the abstract method `existsAllergy` (not shown here) for checking whether the `designation` of an allergy has already been stored in `as` or not. Therefore, if the designation has already been stored, the list of allergies remains unchanged, `as.equals(\old(as))`, otherwise the allergy designation is stored at the end of the list, `as.equals(\old(as).insertBack(desigRepr(-designation)))`. JML abstract method `desigRepr` (not shown here) maps an array of bytes to a unique value. JML only allows side-effects free methods within specifications. This is enforced by using the JML keyword `pure`. All the methods used within specifications in our examples are `pure`, e.g. `existsAllergy` and `insertBack` (which uses an auxiliary `clone()` method) in `addAllergy`.

```
//@ model instance JMLEqualsSequence as;
//@ model instance short size;
```

[3] JML also provides `ghost`, a more limited variation of `model` variables.

```
//@ model instance short maxsize;

/*@ public normal_behavior
  @  requires size < maxsize;
  @  requires designation != null && date != null;
  @  requires existsAllergy(designation);
  @  ensures as.equals(\old(as));
  @    also
  @ public normal_behavior
  @  requires size < maxsize;
  @  requires designation != null && date != null;
  @  requires !existsAllergy(designation);
  @  ensures as.equals(\old(as).insertBack(
  @                       desigRepr(designation)));
  @*/
public abstract void addAllergy ( byte[] designation,
                          byte[] date)
              throws RemoteException, UserException;
```

Abstract specifications are related to actual Java code through the use of the JML `represents` clause. Hence, `as`, declared in `Allergies`, is related to code in the `Allergies_Imp`, which implements `Allergies`. The abstract variable `size` is represented as the concrete field `nextFree`, and `maxsize` as the static variable `MAX_ITEMS`. The pure method `allergiesRepr` represents `as` as a `JMLEquals-Sequence` produced by the insertion of all the elements in `allergies`. In JML, `pure` methods are side-effect free methods.

```
//@ represents size <- nextFree;
//@ represents maxsize <- MAX_ITEMS;

//@ represents as <- allergiesRepr();

/*@ pure model JMLEqualsSequence allergiesRepr() {
  @  JMLEqualsSequence r = new JMLEqualsSequence();
  @    for (short i=0; i < nextFree; i++) {
  @      r = r.insertBack((Object)(allergies[i]));
  @    }
  @  return r;
  @ }
  @*/
```

JML Class and System Invariants The Class invariant CI1 is expressed as the JML invariant below. This invariant is declared in class `Allergy`.

```
//@ instance invariant des.size == CODE_LENGTH;
```

The System invariant SI1 is expressed as the JML invariant below. This invariant suggests that a global access to medicines and appointments in the card must exist. Following the Java Card Remote Method invocation (JCRMI)

approach for communication, in which the Java Card applet is the server, the HealthCard application defines an interface `CardServices` that declares all the services available for remote objects. Class `CardServices_Imp`, an implementation of this interface in Java, accesses the information and the state of any remote object in the card. `CardServices_Imp` declares two variables `med` and `app` for keeping track of medicines and medical appointments respectively. Method `getData()` returns an array of objects of type `Medicine`. Method `getApp()` returns an array of objects of type `Appointment`.

```
/*@ invariant
  @ (\forall int i; i<med.getData().length & i>=0;
  @  (\forall int k; k<app.getApp().length & k>=0;
  @     med.getData()[i].getAppID() ==
  @                    app.getApp()[k].getID()
  @       ==>
  @     med.getData()[i].getDate() >=
  @                    app.getApp()[k].getDate() ))
  @*/
```

3.7 Validation and Verification

We used the JML Common Tools suite [3] to check our implementation of the HealthCard. This suite provides support to the run-time assertion checking of JML specifications. Checking an application with this suite is an iterative process of checking the implementation with respect to the JML specifications, and then evolving either the specification or the implementation (or both) when a run-time error is produced. Errors can be detected before a concrete implementation for the application is written. For instance, Java interfaces and Java abstract classes are checked against JML specifications before writing full implementations for those interfaces and abstract classes. At this point, programmers can go back to an earlier development phase, e.g. modifying some informal functional requirements; thereafter JML specifications are evolved accordingly.

3.8 A JFly Prototype Tool

We have built a prototype tool that automates the process of writing JML formal specifications from simple semi-formal specifications. The prototype tool builds on the idea that semi-formal specifications can be written as requirements of the form if <event> then <restriction>. Therefore, after if and before end, a method precondition exists; and after then a method postcondition occurs. The tool can be reached at `http://www.knowmydream.com/Projects/jfly/`. This is just a prototype tool; it demonstrates how our ideas on JML-based strategy for incorporating formal specifications to the software development of programs can be automated. For instance, the prototype tool transforms the specification if <date is NOT EQUAL TO null AND date's length is EQUAL TO date_model's length> then <date is EQUAL TO date_model>.

```
/*@ public normal_behavior
  @  requires date != null &&
  @           date.length == date_model.length;
  @   ensures date == date_model;
  @*/
```

4 Conclusion

We propose a strategy for evolving informal requirements into formal specifications as part of a software engineering methodology. However, evolving informal requirements into formal specifications is not a linear process: it requires great ingenuity and experience. Although we presented our ideas through the development of a smart card application, we consider that our strategy is suitable for developing correct applications that implement a client-server architecture with a need of a light-weight server specification in general. In this client-server setting, validations of methods' pre-conditions are not carried out within methods' implementations. It is the client's responsibility to ensure that methods are called with the right parameters. This reduces the size of implemented methods. This is particularly important for smart cards whose generated byte-code cannot be bigger than a certain limit to be installed on the smart card. We used JML tool machinery to check that the methods are always called with the right parameters throughout the whole application. This prevents programmers from making validations both inside and outside methods, a common programming mistake. Yet, we chose JML as the formal specification language, our ideas can also be adapted to the development of C++ programs, with formal specifications written in the ACSL (ANSI/ISO C Specification Language) [1] language instead, and the verification work accomplished with the Frama-C Tool [7].

We want to emphasise the importance of thinking of invariant properties when developing software. Thinking about invariants prior to writing code is a practice to which programmers do not easily adhere. Having a formal specification of an application and systematically using a tool, i.e. the JML Common Tools, for checking the correctness of the code as it is written forces programmers to think about how the written code affects the consistency and the correctness of the whole program. It is our experience that invariants are the key notion in formal software development that makes a difference with respect to traditional (non formal methods based) software engineering methodologies [4]. In general, programmers feel intimidated by the idea of coming up with an invariant. Often, they design code that can make their programs be in an inconsistent state. We strongly believe JML helps in this sense, from furnishing a friendly Java-like syntax, to making it possible to use first-order logic predicates in JML specifications naturally.

Finally, the HealthCard application consists of 20 interfaces, 16 concrete classes, and 174 KB in total, with about 4300 lines of code, of which 50% are specifications, 42% are code, and 8% include both specifications and code. The whole development can be reached at `https://sourceforge.net/projects/-healthcard/`. It took about 4 months time to the second and third authors to

write the HealthCard, supervised by the first author, who has a large experience with JML.

References

1. P. Baudin, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto. ACSL: ANSI/ISO C specification language. `http://frama-c.cea.fr/download/-plug-in_development_guide.pdf`.
2. C. Breunesse, N. Catano, M. Huisman, and B. Jacobs. Formal methods for smart cards: An experience report. *Science of Computer Programming*, 55(1-3):53–80, March 2005.
3. L. Burdy, Y. Cheon, D. Cok, M. Ernst, J. Kiniry, G. T. Leavens, K. R. M. Leino, and E. Poll. An overview of JML tools and applications. *International Journal on Software Tools for Technology Transfer (STTT)*, 7(3):212–232, June 2005.
4. N. Catano, F. Barraza, D.García, P. Ortega, and C. Rueda. A case study in JML-assisted software development. In P. Machado, A. Andrade, and A. Duran, editors, *Brazilian Symposium on Formal Methods (SBMF)*, pages 5–21, August 2008.
5. N. Catano and T. Wahls. Executing JML specifications of java card applications: A case study. In *24th ACM Symposium on Applied Computing, Software Engineering Trac (SAC)*, Waikiki Beach, Honolulu, Hawaii, March 8-12 2009.
6. E. W. Dijkstra. *A Discipline of Programming*. Series in Automatic Computation. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1976.
7. The Frama-C Tool. `http://frama-c.cea.fr`.
8. M. Fraser, K. Kumar, and V. K. Vaishnavi. Strategies for incorporating formal specifications in software development. *Communincations of ACM*, 37(10):74–86, 1994.
9. M. G. Ilieva and O. Ormandjieva. Automatic transition of natural language software requirements specification into formal presentation. In *Applications of Natural Language to Information Systems (NLDB)*, pages 392–397, 2005.
10. R. A. Kemmerer. Integrating formal methods into the development process. *IEEE Software*, 7(5):37–50, 1990.
11. G. Leavens, A. Baker, and C. Ruby. Preliminary design of JML: A behavioral interface specification language for Java. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–38, 2006.
12. J. Link. *Unit Testing in Java*. Morgan Kaufmann, 2003.
13. B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, October 1992.
14. B. Meyer. *Object Oriented Software Construction*. Prentice Hall PTR, 1997.
15. L. Shaoying, A. Offutt, C. Ho-Stuart, Y. Sun, and M. Ohba. SOFL: a formal engineering methodology for industrial applications. *IEE Transactions on Software Engineering*, 24, 1998.
16. V. T. Vasconcelos, I. Nunes, and A. Lopes. Monitoring java code using ConGu. In *19th International Workshop on Algebraic Development Techniques (WADT)*. Universit di Pisa, 2008.

Snapshot Isolation Anomalies Detection in Software Transactional Memory

Ricardo J. Dias, João Costa Seco, and João M. Lourenço*

CITI — Departamento de Informática,
Universidade Nova de Lisboa, Portugal
{rjfd,joao.seco,joao.lourenco}@di.fct.unl.pt

Abstract. Some performance issues of transactional memory are caused by unnecessary abort situations where non serializable and yet non conflicting transactions are scheduled to execute concurrently.
Smartly relaxing the isolation properties of transactions may overcome these issues and attain considerable performance improvements. However, it is known that relaxing isolation restrictions may lead to runtime anomalies. In some situations, like database management systems, developers may choose that compromise, hence avoiding anomalies explicitly. Memory transactions protect the state of the program, therefore execution anomalies may have more severe consequences in the semantics of programs. So, the compromise between a relaxed isolation strategy and enforcing the necessary program correctness is harder to setup.
The solution we devise is to statically analyse programs to detect the kind of anomalies that emerge under snapshot isolation. Our approach allows a compiler to either warn the developer about the possible snapshot isolation anomalies in a given program, or possibly inform automatic correctness strategies to ensure Serializability.

Keywords: Snapshot Isolation, Serializable Anomalies, Software Transactional Memory, Static Analysis

1 Introduction

Concurrent programming is becoming mainstream due to the widespread use of multicore processors. Locks are an effective but low-level mechanism to control concurrent threads of execution in such systems, and there is a clear demand for more abstract programming mechanisms.

Concurrency control in Database Systems is achieved by using transactions, that usually comply with the standard properties of atomicity, consistency, isolation and durability (ACID). To achieve increased performance, transactional

* This work was partially supported by Sun Microsystems and Sun Microsystems Portugal under the "Sun Worldwide Marketing Loaner Agreement #11497", by the Centro de Informática e Tecnologias da Informação (CITI), and by the Fundação para a Ciência e Tecnologia (FCT/MCTES) in the Byzantium research project PTDC/EIA/74325/2006, research grant SFRH/BD/41765/2007, and the Streamline research project PTDC/EIA-CCO/104583/2008.

frameworks sometimes relax those properties and allow transactions to execute under more relaxed isolation levels [17,1]. In particular, databases frequently provide the developer the ability to choose among different isolation levels.

More relaxed isolation levels naturally lead to increased overall performance of the transactional system, but also to the triggering of transactional anomalies, such as dirty and unrepeatable reads. Serializability is the strongest isolation level. Snapshot Isolation (SI) is a relaxed isolation level that also avoids anomalies such as unrepeatable and dirt reads. However, SI still allows some other transactional anomalies such as *write skew* and *SI read-only* [7]. Many database applications are known to execute correctly under SI, thus making it a good compromise between correction—which concurrency anomalies are admitted and how do they affect the applications—and performance.

Transactional Memory (TM) was proposed as an alternative programming abstraction for concurrency control in multithreaded programs [16,11]. TM frameworks typically operate in full serializable mode and do not allow one to relax the isolation level. Thus, the potential of Snapshot Isolation for performance improvement, a *de facto* standard for the database world, has been neglected in TM programming until now. Figure 1 illustrates the potential of such improvement by means of a small experiment with a transactional memory benchmark executed in a Sun Fire x4600 with 16 cores. The benchmark operates over a linked list, executing insert, delete and lookup operations. In this example, one can observe that while the Serializable version does not scale under the increasing number of processors/threads, while the Snapshot Isolation version scales almost linearly with the number of processors/threads (note that the scale in the X-axis is logarithmic).

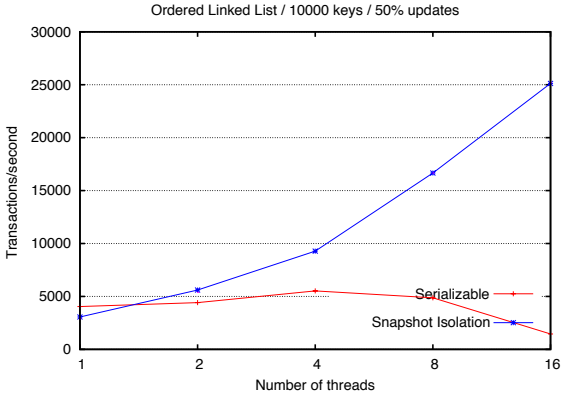


Fig. 1. Snapshot Isolation vs. Serializability in Transactional Memory

The above example is a strong motivation to further study how to use relaxed isolation levels, and in particular Snapshot Isolation, in the transactional memory setting. Unlike the database approaches where a domain specific language (SQL)

is used to model database accesses, TM programs are usually defined in a general purpose programming language, and there is no evidence that there exists a large set of applications that will also execute correctly under weaker TM isolation levels without serious rewriting.

This work aims at asserting that a multithreaded transactional memory program will not trigger the well known SI anomalies when executing under Snapshot Isolation, thus leading to non-serializable executions. In this case, the application will execute as if under Serializable isolation level. Our work grounds in previous work in static detection of SI anomalies in databases [7], but our approach targets the very different domain of Transactional Memory.

As a testbed for our work, we defined a simple imperative language, with no support for pointers. Each transaction is an instance of a program written in this language. We perform a data-flow analysis over each program, extracting the information necessary to detect if the concurrent execution of a set of transactions will generate a serializable anomaly. If the analysis detects no serializable anomalies, than the application will execute correctly. In the opposite, if serialization anomalies are found, they should be considered as possible anomalies and confirmed by other means, as our analysis allow for false positives.

The main contributions of this work include:

- A new data-flow analysis to extract information from transactional programs. This analysis will extract compact read- and write-sets in order to define static dependencies between programs.
- The definition of static dependency between transactional programs using the information retrieved from the static analysis.
- A version of the algorithm proposed in [7] to detect SI anomalies, adapted and optimized for the TM setting. The algorithm will operate over a graph of static dependencies between programs, and will determine if the execution of such programs under SI will be serializable.

The rest of the paper is organized as follows: Section 2 describes the Snapshot Isolation model and the definition of serializable anomalies using static dependencies between programs. Section 3 describes the data-flow static method to gather information about read and write accesses in transactional programs, the procedure to generate the static dependencies using this information, and the algorithm to detect serializable anomalies in the static dependency graph. Section 4 discusses the relations among our work and the related ones. Finally, Section 5 presents some concluding remarks and discusses our plans of evolution of this work.

2 Snapshot Isolation

Snapshot Isolation [1] is a weaker isolation level than Serializable where each transaction performs its read operations in a private snapshot of the state, taken in the beginning of the transaction. All write operations performed by the transaction are stored in a local buffer. All read operations on data items previously written by the transaction are performed from its local buffer.

Considering that the lifetime of a successful transaction is the time span that goes from the moment it starts $start(T_i)$ until the moment it commits $commit(T_i)$. Two successful transactions T_1 and T_2 are said to be concurrent if:

$$[start(T_1), commit(T_1)] \cap [start(T_2), commit(T_2)] \neq \emptyset \quad (1)$$

The write operations of a transaction T_i are not visible to the remaining concurrent transactions. When a transaction T_i is ready to commit, it obeys the *First-Commiter-Wins* rule, which states that it can successfully commit only if there is not a concurrent transaction T_k $(i \neq k)$ which has committed write operations to some item that T_i is also changing. This means that if there are two concurrent transactions updating the same data item, only the first one to commit will succeed.

Snapshot Isolation has some advantages over the Serializable isolation level. By always reading from a snapshot, read-only transactions will never abort. A read-only transaction T_i only sees committed results before $start(T_i)$. Also, read-only transactions will never make read-write transactions to abort.

Snapshot Isolation sounds very appealing, however its application may lead to non serializable executions. These executions result in consistency anomalies that may happen when using Snapshot Isolation [7], namely the *write-skew* and *SI read-only* anomalies.

2.1 Static Isolation Anomalies

Other works have defined serializable anomalies under Snapshot Isolation in terms of database program dependencies [7]. In this work we use the same static dependency definition and adapt it for software transactional memory programs.

The SI anomalies can be described formally using static dependencies between transactional programs. Two transactional programs have a static dependency between them if both programs access the same data item and at least one of them performs a write access. Four types of static dependencies are defined in [7]:

- $P_i \xrightarrow{x-ww} P_j$: The transaction resulting from the execution of program P_i writes data item x and commits, and the transaction resulting from the execution of program P_j also writes data item x and commits.
- $P_i \xrightarrow{x-wr} P_j$: The transaction resulting from the execution of program P_i writes data item x and commits, and the transaction resulting from the execution of program P_j reads data item x, written by P_i, and commits.
- $P_i \xrightarrow{x-rw} P_j$: The transaction resulting from the execution of program P_i reads data item x and commits, and the transaction resulting from the execution of program P_j writes data item x, read by P_i, and commits, and programs P_i and P_j are not concurrent.
- $P_i \overset{x-rw}{\Rightarrow} P_j$: The transaction resulting from the execution of program P_i reads data item x and commits, and the transaction resulting from the execution of program P_j writes data item x, read by P_i, and commits, and programs P_i and P_j are concurrent.

The first three dependencies are said to be *non-vulnerable* dependencies and the last one is said to be a *vulnerable* dependency. Using these dependencies we can build a *Static Dependency Graph* [7] (SDG) where programs correspond to nodes and static dependencies correspond to edges.

The relation between the unsatisfiability of the Serializable property and static dependencies between programs can be signalled by the existence of certain kinds of dangerous structures in the SDG of an application.

Fekete et al. [7] defines the concept of *dangerous structure* in a static dependency graph. He shows that if some $SDG(\mathcal{A})$ has a dangerous structure then there are executions of application $\mathcal{A}$ which may be not serializable, and that if a $SDG(\mathcal{A})$ does not have any dangerous structure then all executions of application $\mathcal{A}$ are serializable.

Definition 1 (Dangerous structures) *We say that a $SDG(\mathcal{A})$ has a dangerous structure if it contains nodes P, Q and R (not necessarily distinct) such that:*

- *There is a vulnerable edge from R to P.*
- *There is a vulnerable edge from P to Q.*
- *Either $Q = R$ or there is a path in the graph from Q to R; that is, (Q, R) is in the reflexive transitive closure of the edge relationship.*

The detection of dangerous structures in a SDG can be performed algorithmically.

We next show how to build an SDG by analyzing the source code of an application and how to detect dangerous structures that point to possible anomalies.

3 Static Analysis

In this section we define a new static analysis procedure for a small imperative language and describe how to build a *Static Dependency Graph* [7] with the information given by the analysis. We next define how to detect execution anomalies.

The following grammar defines the abstract syntax of an imperative language:

$$
\begin{array}{lcl}
\langle E \rangle & ::= & x \mid n \mid \langle E \rangle \; op \; \langle E \rangle \mid \textbf{true} \mid \textbf{false} \mid \textbf{not} \; \langle E \rangle \\
\langle S \rangle & ::= & x := \langle E \rangle \mid \textbf{skip} \mid \langle S \rangle \; ; \langle S \rangle \\
 & \mid & \textbf{if} \; \langle E \rangle \; \textbf{then} \; \langle S \rangle \; \textbf{else} \; \langle S \rangle \mid \textbf{while} \; \langle E \rangle \; \textbf{do} \; \langle S \rangle \\
\langle P \rangle & ::= & \langle S \rangle
\end{array}
$$

This language has integer (n), boolean literals (**true** and **false**), and variables (x). It contains the usual binary arithmetic, logic, and relational operations ($E_1 \;\; op \;\; E_2$). The statements of the language include the conditional and loop statement as well as the variable assignment. We consider that an application is a set of programs defined over a set of shared variables, and each program corresponds to a single memory transaction. We apply the analysis separately to each program.

3.1 Read-Write Analysis

In order to define static dependencies between programs we need to know which data items are read or written by each program that comprises an application. Thus, we use a standard data-flow static analysis to obtain the set of variables read or written by a program. For that purpose we have defined a custom lattice and the appropriate transfer functions.

We establish a state for each shared variables for each node in the control-flow graph of a program. The state of a shared variable is a pair of values of the set $\Gamma = \{?, M, m, \top\}$. The first component of the pair indicates if a variable was read—its read state—and the second component of the pair indicates its write state. A "?" value in the read/write state for a variable x means that x is not read/written by the program. A "M" value in the read/write state for a variable x means that x is indeed read/written by the program. A "m" value in the read/write state for a variable x means that x may be read/written by the program (it is read/written in at least one possible execution path, but not in all). In Figure 2 is depicted the relation order of the lattice Γ.

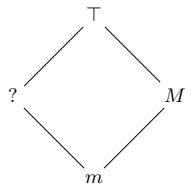


Fig. 2. Lattice Γ order relation diagram.

We now define the data-flow transfer functions over a lattice defined over the set $\Upsilon = \Gamma \times \Gamma \times Var$. The elements of the tuples denote the read state the write state and a shared variable (Var). An element of set Υ of the form $(M, m)_x$ means that variable x is read in every possible execution of the program and is written at least in one possible execution of the program (but not all).
The transfer functions Z_{en} and Z_{ex} map the labels of a control flow to Υ: $Z_{en}, Z_{ex} : Label \longrightarrow \Upsilon$.

Labels are identifiers of the program nodes in the control flow graph (CFG). Each node in the CFG as an entry and an exit point, and functions Z_{en} and Z_{ex} correspond to the entry and exit points respectively.

We define the analysis as a backward procedure by the following functions Z_{en} and Z_{ex}:

Definition 2 (Exit function)

$$Z_{ex}(l) = \begin{cases} \{(?,?)_x \mid x \in FV(P)\} & \text{if } l = final(P) \\ \sqcap\{Z_{en}(l') \mid (l', l) \in flow^R(P)\} & \text{otherwise} \end{cases} \quad (2)$$

where P represents the program we are analyzing, and $final(P)$ denotes the final label that program. $flow^R(P)$ denotes the set of reversed edges of the CFG. The operator $\sqcap$ denotes the greatest lower bound of the sets given by Z_{en}. In the beginning of the analysis, the subset of Υ is initialized with all variables belonging to the set of *Free Variables* and their state is set to $(?, ?)$ (not read nor written). The second entry of this function uses the greatest lower bound operator $\sqcap$ to join the information from more than one node, e.g., the `then` and `else` branches. Because we are using backward analysis, the beginning corresponds to the final label of the program.

Before we present the definition of the Z_{en} function we need to define a binary operator $\oplus$ which is used to modify the read/write state of a variable in a subset of Υ, $(\oplus : \Upsilon \times \Upsilon \longrightarrow \Upsilon)$: For all sets $U, V \subseteq \Upsilon$, $U \oplus V = U \setminus \{(t,s)_x : (t,s)_x \in U \wedge x \in Vars(V)\} \cup V$. Where $Vars(V)$ denotes the set of the variables present in V. The intuition of the operator $\oplus$ is: given $U \subseteq \Upsilon$, which corresponds to the original set, and $V \subseteq \Upsilon$ which corresponds to the set with the modified elements, the $U \oplus V$ operation will remove the elements of U which are also in V that have the same variable, and will return the set of the unmodified values of U with the values of V. This operator is used to override the read/write state of a variable by removing the previous information (in U) and adding the new information (in V). Now we will define the Z_{en} function:

Definition 3 (Entry Function)

$$Z_{en}(l) = \begin{cases} Z_{ex}(l) \oplus \{(M, \hat{\sigma}_y(Z_{ex}(l)))_y : \forall_y \in FV(E)\} & \\ \quad \oplus \{(\bar{\sigma}_x(Z_{ex}(l)), M)_x\} & \text{if } B^l = [x := E], \text{ where } B^l \in blocks(P) \\ Z_{ex}(l) \oplus \{(M, \hat{\sigma}_y(Z_{ex}(l)))_y : \forall_y \in FV(E)\} & \text{if } B^l = [E], \text{ where } B^l \text{ is an elementary block with label } l \\ Z_{ex}(l) & \text{otherwise} \end{cases} \tag{3}$$

Functions $\bar{\sigma}/\hat{\sigma}$ denote the value of the read/write state of a single variable $(\bar{\sigma}, \hat{\sigma} : Var \times \Upsilon \longrightarrow \Gamma)$. The $FV(P)$ function represents the set of free variables in program P.

The first case of Definition 3 introduces the modifications to the read/write state caused by an assignment block, where we change the write state of the assigned variable to M. We also change the read state of all free variables on the right side of the assignment to value M. The second case of the definition treats the evaluation of an expression where we change the read state of all the expression's free variables to value M. The last case in the definition corresponds to the unmodified propagation of the read/write state.

3.2 Generating Static Dependencies

If we consider an application as a set of programs that maybe launched in parallel, and given the read/write set analysis for all those programs, as described in Sect. 3.1, we can compare the set of read and write states of each program with all the other programs and create a Static Dependency Graph (SDG). Since we do not know *a priori* what programs will execute in parallel we pessimistically assume that all programs are concurrent even with several instances of themselves. Building a SGD graph requires $\binom{n}{2} + n$ comparisons. If we consider a large number of programs running in parallel, this may become unbearable. Other techniques can be used, such as the May-Happen-in-Parallel Analysis by Duesterwald and Soffa [5], to determine which programs will execute in parallel and hence help reducing the complexity of the graph construction procedure.

For all pair of programs (P_i, P_j) we compare the two corresponding read-/write states (subsets of Υ resulting from the R/W analysis) and produce a static dependency in the graph. The kind of dependency created depends on the read/write state and also varies if the two programs are concurrent or not. We say that two programs P_i and P_j are not concurrent if they have a write state M for the same variable. By the *First-Commiter-Wins* rule the execution of these two programs is always synchronized and if they run in parallel, one will necessarily abort.

There is a dependency relation between two programs if both access at least one shared data item that is modified by at least one of those programs. Static dependencies are defined from the analysis as follows:

Definition 4 *[Static Dependencies]*

For all programs P_i, P_j in an application, and with the read/write states $U_i, U_j \subseteq \Upsilon$ where U_i is the read/write state of P_i and U_j is the read/write state of P_j. If there is a variable x such that $(\_, w)_x \in U_i$ and $(r, \_)_x \in U_j$ where $w \neq\ ?$ and $r \neq\ ?$ then:

1. *if $(\_, \alpha)_x \in U_i$ and $(\_, \beta)_x \in U_j$ where $\alpha \neq\ ?$ and $\beta \neq\ ?$ then $P_i \xrightarrow{ww} P_j$*
2. *if $(\_, \alpha)_x \in U_i$ and $(\beta, \_)_x \in U_j$ where $\alpha \neq\ ?$ and $\beta \neq\ ?$ then $P_i \xrightarrow{wr} P_j$*
3. *if $(\alpha, \_)_x \in U_i$ and $(\_, \beta)_x \in U_j$ where $\alpha \neq\ ?$ and $\beta \neq\ ?$, and P_i and P_j are not concurrent then $P_i \xrightarrow{rw} P_j$*
4. *if $(\alpha, \_)_x \in U_i$ and $(\_, \beta)_x \in U_j$ where $\alpha \neq\ ?$ and $\beta \neq\ ?$, and P_i and P_j are concurrent then $P_i \overset{rw}{\Rightarrow} P_j$*

It is important to note that comparing two programs P_1 and P_2 implies comparing them in both directions. Most of the times this comparison generates dependencies in both ways. For instance, if we consider programs P_1 and P_2 such that $U_1 = \{(M, m)_x\}$ and $U_2 = \{(m, M)_x\}$, if we compare U_1 with U_2 there is a dependency $P_1 \xrightarrow{wr} P_2$ because P_1 may write variable x that is later read by P_2, and there is also a dependency $P_2 \xrightarrow{wr} P_1$ because P_2 may write variable x which may be read by P_1.

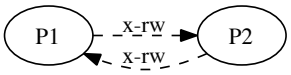


Fig. 3. An *SDG* with a cycle of read-write dependencies for the same variable.

Note that we generate non-vulnerable dependencies if we know that programs P_i and P_j are not concurrent. Otherwise we generate vulnerable dependencies between them.

As an example, consider the program P with state $U = \{(M, m)_x, (?, m)_y\}$ resulting from its data flow analysis. If we compare program P with itself the dependencies generated according to Definition 4 are:

- $P \xrightarrow{ww} P$: several instances of P write variable x or y.
- $P \xrightarrow{wr} P$: P may write variable x, which is read by another instance of P.
- $P \overset{rw}{\Rightarrow} P$: P reads variable x which may be written by another instance of P.

Filtering False Positives When traversing the edges in a *SDG*, consider the situation presented in Figure 3 where there are two vulnerable read-write dependencies for the same variable x forming a cycle between P_1 and P_2. In this case, the application of Definition 1 would identify this as a dangerous structure. Note that P_1 has an incoming vulnerable edge from P_2 and has an outgoing vulnerable edge to P_2, and there is (null length) path cyclic from P_2 to itself.

Now, consider an execution H with two transactions T_1 and T_2, result of the execution of P_1 and P_2 respectively. We argue that is not possible to define an execution H under Snapshot Isolation if there is a transactional dependency $T_1 \xrightarrow{x-rw} T_2$ and a transactional dependency $T_2 \xrightarrow{x-rw} T_1$. Consider that T_1 is executing concurrently with T_2 and there is a dependency $T_1 \xrightarrow{x-rw} T_2$ which states that T_1 reads variable x and T_2 writes variable x, and there is also a dependency $T_2 \xrightarrow{x-rw} T_1$ which states that T_2 reads variable x and T_1 writes variable x. This means that T_1 and T_2 are concurrent, that both write to variable x, and that both commit. However, by the *First-Committer-Wins* rule, one of the two transactions should have aborted, hence it is not possible to define such an execution. This incompatibility between edges also applies to the other kind of dependencies.

Given these observations, if we apply the definition of dangerous structures (Definition 1) to the *SDG* of Figure 3 and follow the edge $P_1 \overset{rw}{\Rightarrow} P_2$ then we can ignore the edge of the same kind for the same variable in the opposite direction, the edge $P_2 \overset{rw}{\Rightarrow} P_1$.

An extension to Definition 1 was made to enable the check for incompatible edges. Algorithm 1 presents the pseudo-code to detect dangerous structures. The *compatible* function will test if an edge e is compatible with the history of edges already visited.

Algorithm 1: Dangerous Structure detection algorithm with incompatibility check.

```
Data: nodes[], edges[], visited[]
Result: true or false
initialization;
foreach Node n : nodes do
    foreach Edge in : incoming(n, edges) do
        if vulnerable(in) then
            add(visited, in);
            foreach Edge out : outgoing(n, edges) do
                if vulnerable(out) and compatible(out, visited) then
                    add(visited, out);
                    if existsPath(target(out), source(in), visited) then
                        return true;
                    end
                end
            end
        end
    end
    clear(visited);
end
return false;
```

4 Related Work

Software Transactional Memory (STM) [16,11] (TM) is a new approach to concurrent programming, promising both, an efficient usage of parallelism and a powerful semantics for concurrency constraint. STM applies the concept of transactions, widely known from the Databases community, into the management of data in main memory. STM promises to ease the development of scalable parallel applications with performance close to finer grain locking but with the simplicity of coarse grain locking.

Memory transactions must only ensure two of the ACID properties: Atomicity and Isolation. The Consistency property is more relaxed as volatile memory does not have a fixed logical structure, like a database system does, over which one can make referential consistency assertions. And the Durability property may be dropped, as memory transactions operate in volatile memory (RAM), a non-persistent data storage.

In the past few years, several STM frameworks have been developed. Most of the STM frameworks take the form of software libraries, providing an API to export the transactional interface to the application [4,2,11,13]. This library-based approach allows the rapid prototyping of algorithms and their performance evaluation. Some other STM frameworks extend existing programming languages with transactional constructs supported directly by the compiler [10,8,14]. Most of these frameworks focus in managed languages such as Java, C#, and Haskell, while some other target unmanaged languages like C and C++.

All the above referenced works implement Serializable isolation to guarantee the correct execution of transactional memory programs. Only [15] implements a STM using Snapshot Isolation called SI-STM. In this work, the authors also proposed a SI safe version where SI anomalies were automatically avoided by the algorithm. Our approach is different since we do not require modifications to existing SI algorithms and we perform a static analysis to assert if a program will execute correctly under SI. To our best knowledge there is no other work in software transactional memory that follows this static approach to detect Serializable anomalies.

The use of Snapshot Isolation in databases is a common place, and there are some works related to the detection do SI anomalies. Our work is clearly inspired in [7]. This work proposes a static analysis methodology for database applications aiming at detecting SI anomalies. Their static analysis was described informally and was applied *ad-hoc* to the database benchmark TPC-C. The work presented in [12] describe a prototype which was already able to analyze database applications automatically, and also presented some solutions to reduce the number of false positives, but shared the theoretical base with [7]. There is another work described in [3] that detect and prevent SI anomalies dynamically with runtime information in database applications.

5 Concluding Remarks

Although static verification of Snapshot Isolation anomalies is not a new topic in database applications, in software transactional memory the use of Snapshot Isolation is much unexplored. The transactional anomalies triggered by the use of SI in transactional memory programs have strong impact in the execution correctness of such programs and is a major drawback to its widespread.

With this preliminary work we show that it may be possible to give stronger guarantees of correct execution under SI which allows to further explore SI algorithms in the context of STMs. We presented a simple data-flow analysis to extract the information of read and write accesses to variables in transactional programs. It gives good results as it does not allow any false negatives but allows some false positives which requires the programmer to verify by hand. Future work will target more complex language with pointers, and also towards techniques to correct programs that are detected to have Serializable anomalies, without changing its semantics and with low impact in their performance.

References

1. Hal Berenson, Phil Bernstein, Jim N. Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. A critique of ansi sql isolation levels. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 1–10, New York, NY, USA, 1995. ACM.
2. João Cachopo and António Rito-Silva. Versioned boxes as the basis for memory transactions. *Sci. Comput. Program.*, 63(2):172–185, 2006.

3. Michael J. Cahill, Uwe Röhm, and Alan D. Fekete. Serializable isolation for snapshot databases. *ACM Trans. Database Syst.*, 34(4):1–42, 2009.
4. Dave Dice, Ori Shalev, and Nir Shavit. Transactional locking ii. In *Distributed Computing*, volume 4167, pages 194–208. Springer Berlin / Heidelberg, October 2006.
5. Evelyn Duesterwald and Mary Lou Soffa. Concurrency analysis in the presence of procedures using a data-flow framework. In *TAV4: Proceedings of the symposium on Testing, analysis, and verification*, pages 36–48, New York, NY, USA, 1991. ACM.
6. K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Commun. ACM*, 19(11):624–633, 1976.
7. Alan Fekete, Dimitrios Liarokapis, Elizabeth O'Neil, Patrick O'Neil, and Dennis Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2):492–528, 2005.
8. Pascal Felber, Christof Fetzer, Ulrich Müller, Torvald Riegel, Martin Süßkraut, and Heiko Sturzrehm. Transactifying applications using an open compiler framework. In *Proceedings of the 2nd ACM SIGPLAN Workshop on Transactional Computing*, August 2007.
9. Tim Harris and Keir Fraser. Language support for lightweight transactions. In *OOPSLA '03: Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications*, pages 388–402, New York, NY, USA, 2003. ACM.
10. Tim Harris, Simon Marlow, Simon Peyton-Jones, and Maurice Herlihy. Composable memory transactions. In *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 48–60, New York, NY, USA, 2005. ACM.
11. Maurice Herlihy, Victor Luchangco, Mark Moir, and III William N. Scherer. Software transactional memory for dynamic-sized data structures. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 92–101, New York, NY, USA, 2003. ACM.
12. Sudhir Jorwekar, Alan Fekete, Krithi Ramamritham, and S. Sudarshan. Automating the detection of snapshot isolation anomalies. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1263–1274. VLDB Endowment, 2007.
13. Dalessandro Luke, Virendra J. Marathe, Michael F. Spear, and Michael L. Scott. Capabilities and limitations of library-based software transactional memory in c++. In *Proceedings of the 2nd ACM SIGPLAN Workshop on Transactional Computing*, Portland, OR, August 2007.
14. Yang Ni, Adam Welc, Ali-Reza Adl-Tabatabai, Moshe Bach, Sion Berkowits, James Cownie, Robert Geva, Sergey Kozhukow, Ravi Narayanaswamy, Jeffrey Olivier, Serguei Preis, Bratin Saha, Ady Tal, and Xinmin Tian. Design and implementation of transactional constructs for c/c++. *SIGPLAN Not.*, 43(10):195–212, 2008.
15. Torvald Riegel, Christof Fetzer, and Pascal Felber. Snapshot isolation for software transactional memory. In *TRANSACT06*, Jun 2006.
16. Nir Shavit and Dan Touitou. Software transactional memory. In *PODC '95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, pages 204–213, New York, NY, USA, 1995. ACM.
17. Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, fifth edition, 2006.

Lightweight Type-Like Hoare-Separation Specs for Java

Tiago Santos

Departamento de Informática FCT/UNL, Lisboa, Portugal
tiago.santos@fct.unl.pt

Abstract. Type systems are effective but not very precise, while program logics tend to be very precise, but undecidable. The aim of this work is extend the expressiveness of more familiar type-based verification towards more informative logical reasoning, without compromising soundness and completeness. We thus investigate a lightweight specification language based on propositional logic for Java and describe a prototype implementation on top of Polyglot. The verification process is modular and based on Dijkstra's weakest precondition calculus, which we extend to a large fragment of the Java object-oriented language. A distinguishing aspect of our approach is a novel "dual" separation logic formulation, which combines Hoare logic with separation logic reasoning in a unified way, allowing us to handle aliasing through a separation of pure from linear properties.

Keywords: Lightweight Specifications, Static Analysis, Verifying Compiler, Hoare Logic, Separation Logic, Weakest Precondition Calculus

1 Introduction

Over the past decades, specification, verification and validation of software present a very important role in software development, since they guarantee correctness and the absence of runtime errors statically, reducing maintenance and development costs. Last year marked the fortieth anniversary of Hoare's article that contributed to the revolution of this subject [1, 2].

The use of formal methods for verifying program properties has witnessed an impulse recently, with tools and programming languages (e.g. ESC/Java2 [3], JACK [4], Spec# [5]) that have great expressiveness power and allow static verification of programs. However, most of them require user interaction and have very complex specification language, which are obstacles for their use.

Lightweight specification languages, on the other hand, thought presenting less expressiveness, still allow reasoning about interesting properties of a system with less effort, thus making its usage compelling in software development.

Figure 1 illustrates a simple example of how to specify the absolute value of a number, ensuring that the result is never negative (`ensures !return:neg`). Other motivating examples are the specification of a buffer, where one can write only if there are free positions and read if the buffer has data; protocols that

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 43–54

```
public int abs(int x)
    ensures !return:neg
{
    if (x > 0) return x;
    else return -x;
}
```

Fig. 1. Specification – Absolute Value of a Number

follow a similar approach, such as specifying an FTP session; and simpler situations, like ensuring that a given variable is not null, avoiding invalid dereferences. With our solution, these checks can be specified in a simple way and with little impact on the compilation process.

This paper presents a lightweight specification language and its integration into the Java programming language, by extending its type system and creating a verifying compiler. This extension, called SpecJava, allows the use of assertions in the Java language, providing developers a way to write correct programs according to their specifications. The specifications can be expressed in a simple way when compared to existing tools and the verification process is fully automatic. The main contributions of this paper are:

- a lightweight specification language for Java and the underlying logic (Sect. 2);
- the technique used to verify a SpecJava program (Sect. 3);
- the implementation of SpecJava (Sect. 4).

2 Lightweight Specification Language

SpecJava's specification language is similar to JML [6] and Spec# [5], but is lightweight and based on a monadic dual logic. It is simpler, not presenting, for example, quantifiers and reference to the value of an expression in its precondition and uses a novel approach to handle aliasing by separating pure from linear properties. Like JML and Spec#, we use the reserved keywords `requires` and `ensures` to describe, respectively, pre and postconditions of a procedure. As for class and loop invariants, we use also the keyword `invariant`.

2.1 Dual Hoare-Separation Logic

In this section we present the underlying logic of the developed specification language denoted dual logic. The purpose of defining a new logic comes from the need of having aliasing control on our specification language, since we are extending an object-oriented language where alias can occur. The original Hoare Logic was designed for an imperative programming language with only simple values, where therefore aliasing does not cause any problems.

As the name suggests, a dual logic formula is composed by two components (Fig. 2): the left side (ϕ), named pure formula, composed by a single formula in propositional logic, states properties of immutable objects, or objects that cannot

$$
\begin{array}{lll}
\psi ::= & \phi + \varphi & \text{(Dual Formula)} \\
\varphi ::= & \emptyset \mid \phi \mid \phi * \varphi & \text{(Linear Formula)} \\
\phi ::= & & \text{(Classic Formula)} \\
& \bot & \text{(Bottom)} \\
\mid & \phi \; lc \; \phi & \text{(Binary Formula)} \\
\mid & \neg\phi & \text{(Negation)} \\
\mid & (\phi) & \text{(Parenthesized Formula)} \\
\mid & P(t_1, t_2, \cdots, t_n) & \text{(Predicate Symbols)} \\
lc ::= & \vee \mid \wedge \mid \Rightarrow \mid \Leftrightarrow & \text{(Logical Connectives)} \\
t ::= & & \text{(Terms)} \\
& c & \text{(Constants)} \\
\mid & x & \text{(Variables)} \\
\mid & f(t_1, t_2, \cdots, t_n) & \text{(Function Symbols)}
\end{array}
$$

Fig. 2. Dual Logic Syntax

be aliased (e.g. primitive types of Java) and the right side (φ), named linear formula, composed by a set of classic formulas in propositional logic, models the linear part of the heap (inspired by separation logic [7]). The formulas in the linear side are disjoint, in the sense that two formulas can not refer to a same linear object, and each formula only talks about a single linear object. Note the existence of predicates and functions that allow, respectively, to express properties (such as relations between two terms for e.g. $>(2,3)$) and compose constants and variables with operators to do certain computations (e.g. $-(2,3)$).

We can also state specific zones of the heap on which we want to mention properties by using heap restriction, defined as follows.

Definition 1 (Heap Restriction). *Let φ be a linear formula and $x_1, x_2, \cdots, x_n$ linear variables, then $\varphi \downarrow \{x_1, x_2, \cdots, x_n\}$ is named restricted linear formula to $x_1, x_2, \cdots, x_n$, that is, the subset of formulas contained in φ that correspond to the variables $x_1, x_2, \cdots, x_n$.*

In addition to this definition, we can exclude parts of the heap over which we do not intend to refer properties.

Definition 2 (Heap Exclusion). *Let φ be a linear formula and $x_1, x_2, \cdots, x_n$ linear variables, then $\varphi - \{x_1, x_2, \cdots, x_n\}$ is named linear formula excluding $x_1, x_2, \cdots, x_n$, that is, the subset of formulas contained in φ that do not contain information of variables $x_1, x_2, \cdots, x_n$.*

As an example, in Fig. 3, we define a linear formula (φ) and apply the two previous definitions.

$$
\begin{array}{c}
\varphi \equiv P_1(x) * P_2(y) * P_3(z) \\
\varphi \downarrow \{x, z\} = P_1(x) * P_3(z) \\
\varphi - \{x, z\} = P_2(y)
\end{array}
$$

Fig. 3. Heap Restriction and Exclusion Example

2.2 Assertions

In this section we present the abstract syntax of SpecJava's assertions. As we can see in Fig. 4, this allows to describe the state in which certain objects or primitive types are, specifically, fields (fn), procedures parameters (pn) and methods return (`return`), or the state of the class itself (`this`). States are composed by a set of basic states, which apply to primitive types. For primitive boolean variables, we associate the states `true` and `false` and for numeric variables, we associate `pos`, `neg` or `zero`. With regard to object references, these can be null references (`null`), or refer to states defined in the class of the object type (sn).

D	::=	CF `+` SLF	(Dual Formula)
SLF	::=	CF | CF `*` SLF	(Sep. Formula)
CF	::=	`true` | `false` | CF bop CF | `!` CF | b : S	(Classic Formula)
bop	::=	`&&` | `||` | `=>` | `<=>`	(Logical Connectives)
b	::=	fn | `this` | `return` | pn	(Properties/States – Target)
S	::=	`true` | `false` | `pos` | `neg` | `zero` | `null` | sn	(Properties/States)

$pn, sn, fn \in$ parameter/state/field names

Fig. 4. Abstract Syntax – Assertions

2.3 Classes

In this section we present the abstract syntax for classes. As we can see in Fig. 5 the novelty is the class specification. An invariant declared as `invariant` D express a property that all classes instances must satisfy. A class preserves its invariants if all methods preserve those invariants. However, contrarily to Spec#, where we can only temporarily break invariants via an explicit statement, in our solution invariants can be broken during a method's execution, as long as they are restored at the end. The constructors must guarantee also, in addition to their postconditions, the invariants of the class.

In addition to class invariants, class level specifications are composed by two other constructions, to define states/properties associated to the class. These can

classDecl	::=	`class` cn { *classMember** }	(Class Declaration)
classMember	::=	... | *field* | *method* | *constructor* | *classSpec*	(Class Member)
classSpec	::=		(Class Specification)
		`define` sn`;`	(Abstract Definition)
		| `define` sn `=` D`;`	(Concrete Definition)
		| `invariant` D`;`	(Class Invariant)
field	::=	T fn (`=` E)? `;`	(Instance Variable)

$mn, cn, sn, fn \in$ method/class/state/field names

Fig. 5. Abstract Syntax – Classes

be concrete or abstract, and are declared as `define` sn = D and `define` sn, respectively. Concrete definitions are built based on states/properties observed in class instance variables and/or other definitions at class level (abstract or concrete). Abstract definitions represent class abstract states/properties, without using other definitions and/or states observed in class instance variables.

2.4 Procedures

In this section we present the abstract syntax for procedures. Methods and constructors specification consists of two formulas concerning to preconditions and postconditions, declared as `requires` D, `ensures` D, respectively (Fig. 6).

method	::=	*modifier* T $mn(\overline{arg})$ $\overline{spec}$ { ST }	(Method Declaration)
constructor	::=	*modifier* $cn(\overline{arg})$ $\overline{spec}$ { ST }	(Constructor Declaration)
modifier	::=	`static` | ... | `pure`	(Modifiers)
spec	::=		(Procedure Specification)
		`requires` D	(Precondition)
	|	`ensures` D	(Postcondition)
ST	::=	...	(Statement)
	|	`assume` D	(Assume)
	|	`sassert` D	(Static Assert)

$mn, cn \in$ method/class names

Fig. 6. Abstract Syntax – Procedures

The preconditions of a procedure specify conditions that must be true at the beginning of its execution. In these conditions we can refer to properties from the class, fields, and method parameters. With respect to postconditions, they designate the object state after performing the operation, and may involve, in their conditions method's return state.

In addition to these specifications, assume and assert statements are also supported, with the usual meaning of assuming or verifying a condition at a given point by using `assume` D and `sassert` D, respectively. Last but not least, to specify that a procedure do not change the state of an object, Java modifiers are extended with the keyword `pure`.

As an example, in Fig. 7 we create a buffer class with the states full and empty (Fig. 7a) and the respective constructor with a specification to ensure that the buffer is created empty (Fig. 7b).

3 Program Verification

This section presents the verification process of a SpecJava program. Our approach is based on the weakest precondition calculus (wp-calculus, Definition 3), initially proposed by Dijkstra [8, 9], that extended Hoare Logic [1] by creating

```
public class Buffer {

   define empty = count:zero;
   define full;

   private int buffer[];
   invariant + !buffer:null;
   ...

   private int count;
   invariant !count:neg;
   ...

}
```

(a) Class Specification

```
public Buffer(int size)
   requires size:pos
   ensures + empty && !full
   ...
{ ...
  assume + empty && !full;
}

public pure int dataSize()
   ensures return:zero
             || return:pos
{  sassert !count:neg;
   return count;
}
```

(b) Procedures Specification

Fig. 7. SpecJava – Buffer Specification Example

a method to define the semantics of an imperative programming language, assigning to each statement a predicate transformer, allowing validity verification of a Hoare Triple. In this work, we propose an extension to that calculus, for an object-oriented language, in this case Java. According to the properties referred in [9, pp. 18:19], to prove that a SpecJava program is correct against its specification, it is necessary to associate to each statement its predicate transformer.

Definition 3 (Weakest Precondition). *Let S be a sequence of statements and R its postcondition, then, the corresponding weakest precondition is represented as:*

$$wp\,(S,\ R)$$

Figures 8 and 9 illustrate the more relevant weakest preconditions rules, which concern to loops, object creation and non-void method invocation.

Figure 8 presents wp-calculus for pure statements. In loops, the loop condition (ε) must be pure, composed only by constants or variables of primitive types. Regarding to object creation, these have to be immutable or pure. In this approach an object is considered pure if all methods of the object class are pure. For non-void method invocation the return value is also pure. We can see that the weakest precondition rule for loop is quite simple and corresponds to its invariant, since the invariant must hold in every iteration of the loop and it also must be valid at the beginning of the loop, corresponding to the premises of the rule.

Concerning to object creation, it is necessary to have as its weakest precondition, on the pure side of the result, in respect to a postcondition C + S the class constructor precondition of the object that we are instantiating, and also that we end in a state whose constructor postcondition implies C. For the linear part of the result, since we can have linear arguments we must assure as weakest precondition the constructor precondition, remaining all the facts of elements

$$\text{[loop]}\quad \frac{(I \wedge \neg\varepsilon) \Rightarrow R \quad (I \wedge \varepsilon) \Rightarrow wp(ST, I)}{wp\left(\begin{array}{l}\texttt{while } (\varepsilon)\\ \quad \texttt{invariant } I \texttt{ , } R\\ ST\end{array}\right) = I}$$

$$\text{[pure creation]}\quad \frac{S{\downarrow}\{\overline{z}\} = \emptyset}{wp(x = \texttt{new } cn(\overline{y},\, \overline{z}),\, C{+}S) = \left(\begin{array}{c} Q_{cn_A}[\overline{p_1}/\overline{y}] \\ \wedge \\ \left(\left(\begin{array}{c} f \neq \texttt{null} \\ \wedge \\ R_{cn_A}[this/f, \overline{p_1}/\overline{y}] \end{array}\right) \Rightarrow C[x/f]\right) \end{array}\right) + \left(\begin{array}{c} Q_{cn_B}[\overline{p_2}/\overline{z}] \\ * \\ S - \{\overline{z}\} \end{array}\right)}$$

$$\text{[pure non-void call]}\quad \frac{S{\downarrow}\{\overline{z}\} = \emptyset}{wp(x = k.mn(\overline{y},\, \overline{z}),\, C{+}S) = \left(\begin{array}{c} k \neq \texttt{null} \wedge Q_{mn_A}[this/k,\ \overline{p_1}/\overline{y}] \\ \wedge \\ \left(\left(\begin{array}{c} k \neq \texttt{null} \\ \wedge \\ R_{mn_A}[this/k,\ \overline{p_1}/\overline{y}, return/f] \end{array}\right) \Rightarrow C[x/f]\right) \end{array}\right) + \left(\begin{array}{c} Q_{mn_B}[\overline{p_2}/\overline{z}] \\ * \\ S - \{\overline{z}\} \end{array}\right)}$$

$Q_{mn}/Q_{cn} \in$ method/constructor precondition
$R_{mn}/R_{cn} \in$ method/constructor postcondition
$\varepsilon \in$ pure expression
$\overline{p_1}/\overline{y} \in$ pure formal/concrete parameters
$\overline{p_2}/\overline{z} \in$ linear formal/concrete parameters
$f \in$ fresh name

Fig. 8. WP Calculus – Pure Rules

that are not affected by the procedure call on the heap, through heap exclusion (cf. Definition 2) of the constructor linear parameters. For non-void method invocation the weakest precondition is similar to object creation, but in addition we must guarantee that we are not doing a null reference invocation ($k \neq$ `null`). We can see also that we replace x and the return variable by a fresh name f in the pure side, because the result is pure and we are assigning a new value to the variable x that corresponds to the newly created object or the result of the method. Last but not least, we also need to have the auxiliary condition in the premise, which allow us to check that the postcondition S does not refer to states of the linear parameters, assuring the linearity of our calculus.

Figure 9 shows weakest precondition calculus for linear statements. As we can see these rules are quite similar to pure statements weakest precondition calculus. For procedures we must exclude variable x from the heap (cf. Definition 2) because now x is linear and we are assigning it a new value that does not exist at the precondition state. For methods we must also guarantee that we exclude information about the object where we are calling the method because since the call is not pure we assume that the state of the object changes. In respect to linear assignment, this is similar to Hoare assignment rule and we must guarantee as precondition that variable y acquire x properties by replacing all occurrences of x by y.

As for the auxiliary conditions in the premises, we must check in object creation and method call that after the invocation we end in a state where the

[linear assign]

$$\frac{S{\downarrow}\{y\} = \emptyset}{wp(x = y,\, C{+}S) = C{+}S[x/y]}$$

[linear creation]

$$\frac{S{\downarrow}\{\overline{z}\} = \emptyset \qquad \texttt{true}{+}\Big(\big(x \neq \texttt{null} \wedge R_{cn_B}[this/f]\big) \Rightarrow S{\downarrow}\{x, \overline{z}\}[x/f]\Big)}{wp(x = \texttt{new}\ cn(\overline{y}, \overline{z}),\, C{+}S) = \left(\begin{array}{l} Q_{cn_A}[\overline{p_1}/\overline{y}] \wedge \\ \big(R_{cn_A}[\overline{p_1}/\overline{y}] \Rightarrow C\big) \end{array}\right){+}\big(Q_{cn_B}[\overline{p_2}/\overline{z}] * S{-}\{x, \overline{z}\}\big)}$$

[linear non-void call]

$$\frac{S{\downarrow}\{\overline{z}\} = \emptyset \qquad \texttt{true}{+}\Big(\big(k \neq \texttt{null} \wedge R_{mn_B}[this/k, return/f]\big) \Rightarrow S{\downarrow}\{k, x, \overline{z}\}[x/f]\Big)}{wp(x = k.mn(\overline{y}, \overline{z}),\, C{+}S) = \left(\begin{array}{l} Q_{mn_A}[\overline{p_1}/\overline{y}] \wedge \\ \big(R_{mn_A}[\overline{p_1}/\overline{y}] \Rightarrow C\big) \end{array}\right){+}\left(\begin{array}{l} \big(k \neq \texttt{null} \ \wedge Q_{mn_B}[this/k,\ \overline{p_2}/\overline{z}]\big) * \\ S - \{k,\ x,\ \overline{z}\} \end{array}\right)}$$

$Q_{mn}/Q_{cn} \in$ method/constructor precondition
$R_{mn}/R_{cn} \in$ method/constructor postcondition
$\overline{p_1}/\overline{y} \in$ pure formal/concrete parameters
$\overline{p_2}/\overline{z} \in$ linear formal/concrete parameters
$f \in$ fresh name

Fig. 9. WP Calculus – Linear Rules

procedure's postcondition implies the linear zone of the heap modified by the procedure, by restricting the heap (cf. Definition 1) and that the postcondition S does not refer to states of the linear parameters. It is still necessary to verify, in linear assignment that we do not have information of variable y on the linear side of the heap, since our calculus is linear and the state of y is transfered to x after the assignment, removing all the facts of y from the heap.

To verify that a program is correct according to its specification, the following Hoare Triples must be valid:

$$\forall_{mn} : \{\ Q_{mn} \wedge I_c\ \}\ ST\ \{\ R_{mn} \wedge I_c\ \}$$

$$\forall_{cn} : \{\ Q_{cn}\ \}\ ST\ \{\ R_{cn} \wedge I_c\ \} \quad ,$$

where ST is the procedure body, composed by a sequence of statements and I_c corresponds to class invariants. Thus, methods must preserve class invariants and constructors in addition to its postconditions must assure also the class invariants.

Supposing that we have a SpecJava program, we want to verify that it is valid against its specification using the above mentioned weakest precondition calculus. The verification process is modular, that is, only one procedure at a time is verified. Considering the body of a procedure of this language as the statements sequence $s_1, s_2, \ldots, s_n$ with precondition $\{\ Q\ \}$ and postcondition $\{\ R\ \}$, by applying wp-calculus rules, we obtain as a final result the more general precondition ($\{\ Q_0\ \}$). After this process, for the program to be valid according to its specification, the precondition of the procedure has to imply the more general one ($Q \Rightarrow Q_0$).

The formulas obtained in the verification conditions generation process are formulas in propositional logic, unlike other situations where are generated formulas in first order logic, due to quantifiers, which are not present in the developed specification language, therefore reducing them to a propositional calculus. A formula in propositional logic is said to be satisfiable if we can assign logical values to its variables so that the formula is true. This boolean satisfiability problem is NP-complete, thought is decidable and can be solved using a SAT-Solver. However, the formulas obtained by our calculus contain, uninterpreted predicates and functions that require specific background theories to be solved. For obtaining solutions to these problems, is common to use SMT-Solvers [10].

Despite of an NP-complete problem, there are finite algorithms for obtaining solutions to these problems. However these algorithms execution time may be too high due to the size of the formulas to be verified. Although, we think that this is not a problem to this solution, since it is modular, procedure by procedure, and the size of the formulas is not very high.

4 Implementation

This section presents some implementation details of our system. The extension of the Java language was implemented using the Polyglot tool [11], which implements an extensible compiler for Java 1.4. This tool is also implemented in Java and, in its simplest form only, performs semantics verification of Java. However, it can be extended in order to define changes in the compilation process, including changes in the abstract syntax tree (AST) and semantic analysis.

Polyglot has been used in several projects and has proved to be quite useful when developing compiler extensions to Java-like languages. This work's specification language was fully integrated in the Java compiler, extending the Java syntax with the language proposed in Sect. 2.

The verification process of a program was integrated into Polyglot compilation passes. In addition we extended the semantics verification and typification with new conditions, like not allowing to state a linear property on a pure formula, properties that are not declared, assuring that primitive types only refer to base properties, etc. We developed an internal representation for propositional and dual logic formulas aiming an independent format that can be used by any SMT-Solver. We have also implemented a visiting architecture over formulas to perform operations needed by wp-calculus, such as variable substitution, conversion to conjunctive normal form, obtaining pure and linear variables of a formula, etc. As for the wp-calculus, it is realized in a new pass, after type checking, that goes through each procedure generating the corresponding verification conditions. After this step another pass translates the generic formulas to the SMT-Solver representation and submits it for validity proof. Thus by using this architecture the verification process is independent from a particular SMT-Solver until the submission point, allowing further extensions to any SMT-Solver just by adding a class that translates the generic formulas to the solver's input format or the recently used SMT-lib format. The current SMT-Solver used is

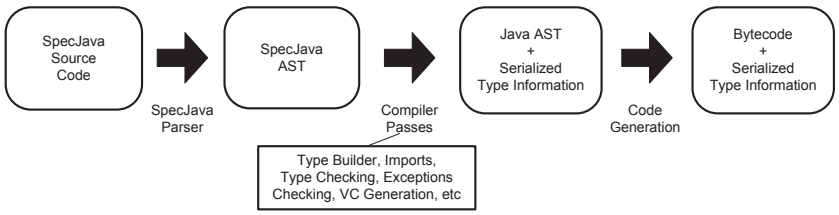


Fig. 10. SpecJava Compiler Architecture

CVC3, it is efficient and has the necessary built-in theories (equality over uninterpreted function and predicate symbols, real and integer linear arithmetic) to prove our verification conditions.

We illustrate the compiler architecture of the developed language in Fig. 10. First, the source code is parsed, generating the corresponding AST. Next, several passes are performed over the AST, including the passes described above. In these passes, if any error occurs (e.g. typing, invalid specification), the compilation process terminates showing the cause of the error, the location in the source code and a counter-example, in case of invalid specification. In a next stage, the AST is translated into a Java AST with type information serialized, preserving new types created by the extension. Finally, the Java code obtained from the previous pass is compiled to bytecode by a standard Java code compiler (e.g. `javac`).

5 Related Work

There are lots of tools and programming languages that support program verification according to its specification (e.g. ESC/Java2 [3], Eiffel [12], Spec# [5]). Some of them, like Eiffel, transform program specification into executable code and perform those verifications at runtime, while others also support formal verification of a program with static analysis (e.g. Spec#). There are four properties that characterize these tools: specification language used, programming language coverage, verification techniques and verification mode.

Regarding the specification language used, tools like ESC/Java2, LOOP [13], JACK [4] and Forge [14] use JML. In KeY [15] specifications are written in OCL. Spec# programming language and jStar [16] tool, have their own specification language. Spec#'s specification language is similar to JML and jStar's is farmost different from the others. In our approach, the specification language is closer to Spec# and JML, but with the novelty of separating pure from linear properties, modeling the heap on the linear side of a formula to track aliasing problems, that cannot be expressed on those languages. Spec#, instead, uses a ownership model to deal with aliasing and specifies frame conditions explicitly by using a `modifies` clause denoting which pieces of the program state a method is allowed to modify.

As for programming language coverage, Spec# and ESC/Java2 cover most of their respective target languages' constructions. In ESC/Java2 it is also possible

to detect synchronization errors such as race conditions and deadlock situations at compile time. Certain tools do not support some features of Java such as dynamic class loading or multithreading (e.g. KeY, Forge, LOOP). In this version of our solution some features of Java are not supported, like inheritance, exceptions, break, continue and switch statements, interface specifications.

With regard to verification techniques, there are several approaches. ESC/-Java2, LOOP, JACK and Spec# use Dijkstra weakest precondition calculus or variants to generate verification conditions. KeY tool uses dynamic logic, where deduction is based on symbolic execution of the program. Forge uses a technique named limited verification that uses symbolic execution and reduces the problem to boolean variables satisfiability. As for jStar, it combines abstract predicate family with symbolic execution and abstraction using separation logic. LOOP defines a denotational semantics in PVS, in contrast to the approaches followed by ESC/Java2, JACK and Spec# that depend directly on an axiomatic semantics. Our approach is based on a variant of Dijkstra wp-calculus to object-oriented languages resembling ESC/Java2, JACK and Spec# and also depends on an axiomatic semantics.

Finally, with regard to verification mode, in ESC/Java2, Forge, jStar and Spec# the verification process is fully automatic, as our SpecJava. LOOP tool needs user intervention. KeY and JACK support both modes.

With these tools we can verify a program against its specification. They have high expressiveness level and allow very complex specification. However, this is the main reason for its rejection by developers, who in general do not have high expertise in logic, nor intend to deal with all the complex mechanisms that are associated with most of these tools. Another point is the fact that most tools are not native to the respective programming language, which forces the use of separated tools in the software development process. On the other hand, our work has less expressive power, but still allows interesting specifications to be written, and its simplicity is appealing to developers.

6 Concluding Remarks and Future Work

In this paper we presented a lightweight specification language for Java, its integration in the compilation process, extending the checks carried out by the type system, and the underlying calculus of the verification process of a SpecJava program. The lightweight specification language developed is closer to JML and Spec#, it is based on propositional logic and is quite intuitive, allowing developers to specify their programs easily and check them automatically at compile time.

This work contributes for a better and easier integration of program verification during its development by augmenting the programming language design. For future work we highlight the following points:

- extend wp-calculus to support inheritance and exception mechanisms;
- support specification at interface level and in separated files, allowing core Java classes specification;

– support `break` and `continue` statements, that change the execution flow of a program.

Acknowledgments. I would like first to thank Prof. Luís Caires for his guidance throughout the development of this work. I thank to Mario Pires and Luísa Lourenço for their comments on previous versions of this paper.

References

1. C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10):576–580, October 1969.
2. C. A. R. Hoare. Retrospective: An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 52(10):30–32, 2009.
3. Cormac Flanagan, K. Rustan M. Leino, Mark Lillibridge, et al. Extended Static Checking for Java. In *PLDI '02: Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, pages 234–245. ACM, New York, NY, USA, 2002.
4. G. Barthe, L. Burdy, J. Charles, et al. JACK – a tool for validation of security and behaviour of Java applications. *Lecture Notes in Computer Science*, 4709:152, 2008.
5. Mike Barnett, Leino, and Wolfram Schulte. *The Spec# Programming System: An Overview*, volume 3362/2005 of *Lecture Notes in Computer Science*, pages 49–69. Springer, Berlin / Heidelberg, January 2005.
6. Gary T. Leavens, Albert L. Baker, and Clyde Ruby. JML: A Notation for Detailed Design, 1999.
7. John C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. *Logic in Computer Science, Symposium on*, 0:55–74, 2002.
8. Edsger W. Dijkstra. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Communications of the ACM*, 18(8):453–457, August 1975.
9. Edsger W. Dijkstra. *A Discipline of Programming.* Prentice Hall, Inc., October 1976.
10. L. De Moura and N. Bjørner. Satisfiability Modulo Theories: An Appetizer. *Formal Methods: Foundations and Applications*, pages 23–36, 2009.
11. Nathaniel Nystrom, Michael R. Clarkson, and Andrew C. Myers. Polyglot: An Extensible Compiler Framework for Java. In *12th International Conference on Compiler Construction*, pages 138–152. Springer-Verlag, 2003.
12. Bertrand Meyer. *Object-Oriented Software Construction.* Prentice Hall PTR, second edition, March 2000.
13. B. Jacobs and E. Poll. Java Program Verification at Nijmegen: Developments and Perspective. *Lecture Notes in Computer Science*, pages 134–153, 2004.
14. G.D. Dennis. *A Relational Framework for Bounded Program Verification.* PhD thesis, Massachusetts Institute of Technology, 2009.
15. Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, et al. The KeY tool. *Software and Systems Modeling*, pages 32–54, April 2004.
16. Dino Distefano and Matthew. jStar: Towards practical verification for Java. *SIGPLAN Not.*, 43(10):213–226, 2008.

Monitorização da Correcção de Classes Genéricas *

Pedro Crispim, Antónia Lopes, and Vasco T. Vasconcelos

Faculdade de Ciências, Universidade de Lisboa,
Campo Grande, 1749–016 Lisboa, Portugal,
{pedro.crispim,mal,vv}@di.fc.ul.pt

Resumo Dado que os genéricos se tornaram muito populares nas linguagens de programação OO, o facto de um método formal não suportar genéricos limita extraordinariamente a sua utilidade e eficácia. De forma a ultrapassar este problema no CONGU, uma abordagem à monitorização da correcção de programas Java face a especificações algébricas, propusemos recentemente uma noção de *mapa de refinamento* que permite definir uma correspondência entre especificações paramétricas e classes genéricas. Baseados nestes mapas, definimos uma noção de correcção de programas face a especificações. Neste artigo, propomos uma forma de monitorizar, em tempo de execução, esta noção de correcção e apresentamos a solução de desenho que suporta este processo na versão 2 da ferramenta CONGU.

1 Introdução

A especificação formal é uma actividade importante no processo de desenvolvimento de software já que, por um lado, auxilia a compreensão e promove a reutilização e, por outro, possibilita a utilização de ferramentas que analisam automaticamente a correcção das implementações face ao especificado. Uma das formas de proceder à analise automática da fiabilidade de componentes de software é através da verificação em tempo de execução (*runtime verification*). Esta abordagem envolve a monitorização e análise das execuções do sistema; à medida que o sistema executa é testada a correcção do comportamento dos componentes relativamente ao que foi especificado.

Apesar dos genéricos se terem tornado muito populares em linguagens como o Java e o C#, as técnicas e ferramentas disponíveis para verificar a correcção de implementações face a especificações não são utilizáveis quando há classes genéricas envolvidas. Este era também o caso do CONGU, uma abordagem à monitorização da correcção de programas Java face a especificações algébricas que temos vindo a desenvolver [7,13]. A ferramenta CONGU que apoia esta abordagem é, há vários anos, intensivamente usada pelos nossos alunos na disciplina de Algoritmos e Estruturas de Dados. Os alunos recebem especificações formais dos tipos de dados que têm de implementar e recorrem à ferramenta para ganhar confiança relativamente à correcção das classes que produzem. Uma vez que os genéricos são extremamente úteis na implementação de tipos de dados em Java, a partir do momento em que passámos a fazer uso dos genéricos na disciplina, o facto do CONGU não suportar genéricos tornou-se um problema. Decididos a ultrapassá-lo, começámos por desenvolver uma noção de *mapa de refinamento*

* Este trabalho foi financiado parcialmente pela FCT através do projecto QUEST, PTDC/EIA-EIA/103103/2008.

que permite definir uma correspondência entre especificações paramétricas e classes genéricas [12]. Baseados nestes mapas, definimos uma noção de correcção de implementações face a especificações mais abrangente. Este trabalho preparou o terreno para a extensão da abordagem CONGU que se apresenta neste artigo. Discute-se ainda a forma como foi concretizada esta extensão na nova versão da ferramenta.

A solução para o problema da monitorização de programas Java que foi desenvolvida de forma a acomodar os genéricos, e que é descrita neste artigo, é substancialmente diferente da usada anteriormente no CONGU [13]. Anteriormente, a estratégia passava por trocar as classes originais por classes *proxy* que usavam classes geradas automaticamente, anotadas com contractos monitorizáveis, escritos em JML [10]. Os principais aspectos inovadores da solução adoptada no CONGU2 são: (i) a introdução de mecanismos que permitem lidar com as classes genéricas e os seus parâmetros e verificar que estão correctas face ao especificado; (ii) o facto de se ter abandonado a geração de classes *proxy* (que traziam problemas quando as classes originais faziam uso de certas primitivas da linguagem Java, como classes internas ou anotações) e se ter passado a utilizar a instrumentação dos binários Java; (iii) o facto de se ter abandonado o JML (o qual não suporta genéricos) e se ter passado a gerar código que, recorrendo a asserções Java, trata directamente da monitorização das propriedades especificadas.

O artigo está estruturado da seguinte forma. A secção 2 fornece uma visão abrangente do CONGU. A secção 3 apresenta a noção de correcção de programas e discute a forma como as propriedades de objectos induzidas pelas especificações podem ser monitorizadas. A solução que foi concretizada no CONGU2 é apresentada na secção 4. O artigo termina apresentando as conclusões na secção 5.

2 Visão geral da abordagem CONGU

O CONGU suporta a monitorização da correcção de programas Java face a especificações algébricas. Nesta secção, recorrendo a um exemplo simples, fornece-se uma visão geral da abordagem, focada essencialmente em aspectos que são visíveis aos seus utilizadores: as especificações, os módulos e os mapas de refinamento.

O exemplo escolhido foram as *listas com fusão*, um tipo de listas (i) cujos elementos são "fundíveis" e (ii) que têm uma operação — mergeInRange — que funde os elementos da lista num determinado intervalo de posições $i...j$. A figura 1 mostra os três elementos envolvidos na especificação deste tipo de dados: ListWM, Mergeable e LWM. ListWM é um exemplo de uma especificação paramétrica, enquanto que Mergeable é um exemplo de uma especificação simples (usada como parâmetro na primeira).

Cada especificação introduz um único género. No exemplo, Mergeable define o género simples com o mesmo nome enquanto que ListWM introduz um género composto ListMW[Mergeable]. O género **int** é primitivo na linguagem.

Cada especificação declara um conjunto de operações e predicados, classificando-os como constructors, observers ou others. As operações classificadas como constructors são aquelas com as quais se podem construir todos os valores do género. As restantes operações e predicados permitem obter informação adicional acerca de valores do género ou oferecem formas alternativas de os construir e, portanto, recebem como argumento um valor do género (por convenção, no primeiro). A classificação das operações

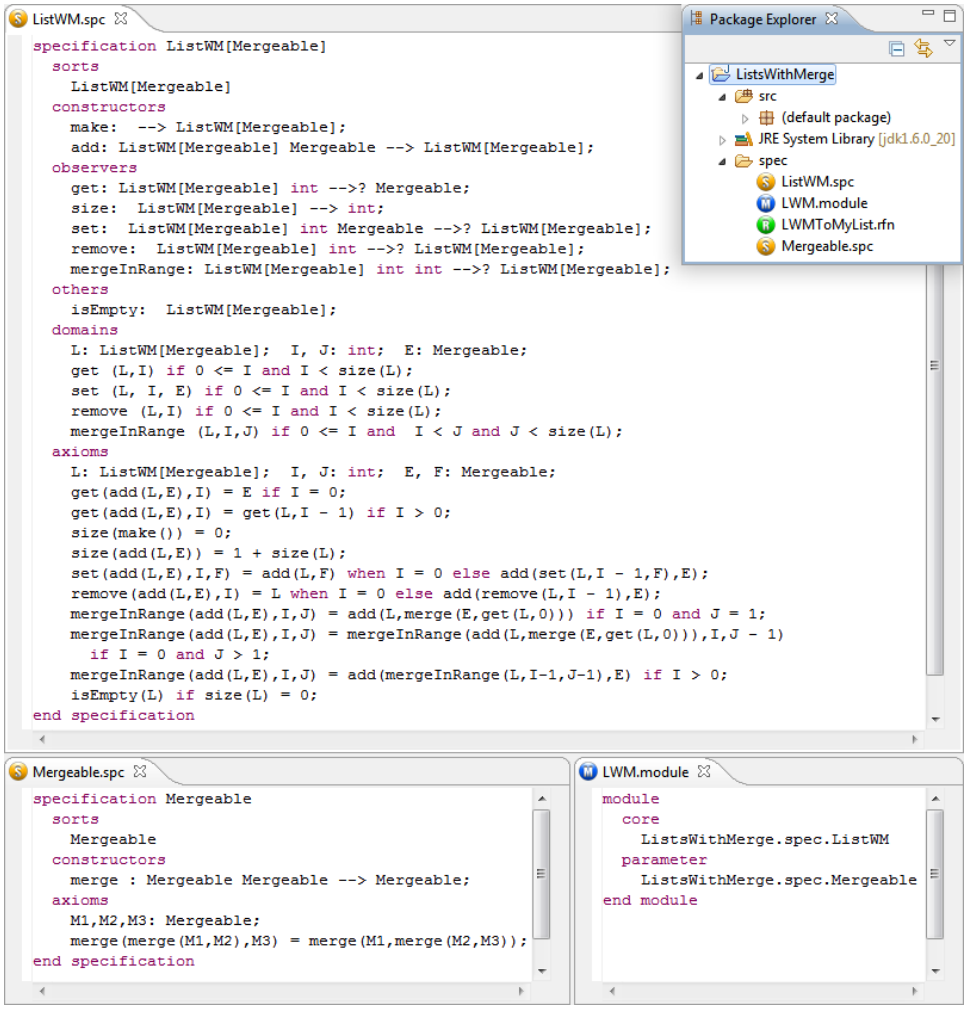


Figura1. Os três elementos envolvidos na especificação do tipo de dados *listas com fusão*.

como observers ou others apenas se reflecte na estrutura sintáctica dos axiomas que podem ser usados para definir as suas propriedades. As propriedades dos observers têm de ser expressas usando construtores no primeiro argumento e variáveis nos restantes, enquanto que no caso dos others podem ser usados variáveis em todos os argumentos.

Porque as operações podem ser parciais, cada especificação define as situações em que operação parcial tem de estar definida — a chamada *condição de domínio*. Por exemplo, em ListWM, a operação get é declarada como sendo parcial e é definido que get(L, I) tem de estar definida se I é um índice da lista L.

A ligação entre diferentes especificações é realizada através de um elemento adicional — os *módulos* (LWM, no nosso exemplo). Nestes módulos, as especificações identificadas como *core* definem os tipos de dados que têm de ser implementados en-

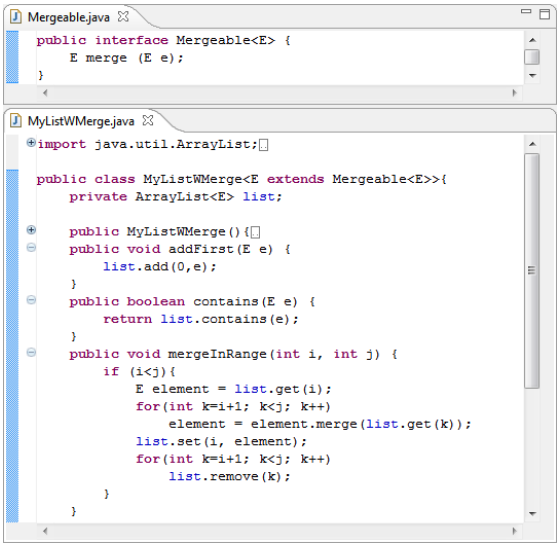


Figura2. The interface `Mergeable<E>` e an excerpt of the Java class `MyListWMerge<E>`.

quanto que as *parameter* não carecem de implementação; servem apenas para impor limitações à instanciação de certos tipos.

Suponha-se que temos uma implementação candidata para o módulo LWM e que gostaríamos de verificar a sua correcção. Primeiro, é necessário estabelecer uma correspondência entre o género definido por cada especificação *core* S de LWM e um tipo Java T definido por uma das nossas classes. Mais ainda, precisamos de estabelecer uma correspondência entre as operações e predicados de S e os métodos e construtores de T. No CONGU, esta correspondência é definida através de um *mapa de refinamento*. Este permite ainda ligar as especificações parâmetro com as variáveis de tipo das classes genéricas: cada género definido por uma especificação parâmetro S é ligada a uma variável de tipo E e cada operação/predicado de S é ligada a uma assinatura de método.

Suponha-se que a implementação candidata para o módulo LWM consiste na classe `MyListWMerge` e no interface `Mergeable` apresentados na figura 2. Na figura 3 é estabelecida uma correspondência entre LWM e esta implementação. Esta faz corresponder o género ListWM[Mergeable] ao tipo definido por `MyListWMerge` e as operações e predicados do primeiro aos métodos e construtores do segundo. Por exemplo, o make é enviado para o construtor `MyListWMerge()` e o add é enviado para o método `addFirst`.

O primeiro argumento de uma operação corresponde sempre ao objecto **`this`** e, portanto, uma operação é enviada sempre para um método com menos um argumento. Apenas as operações declaradas como constructors podem ser enviadas em construtores Java. Os predicados são sempre enviados para métodos cujo tipo de retorno é **`boolean`**. As operações que produzem elementos do género definido pela especificação são enviadas para métodos que tanto podem ser **`void`** como retornar um elemento do tipo correspondente. Desta forma é possível considerar tanto implementações com objectos mutáveis como imutáveis. No nosso exemplo, a classe `MyListWMerge` for-

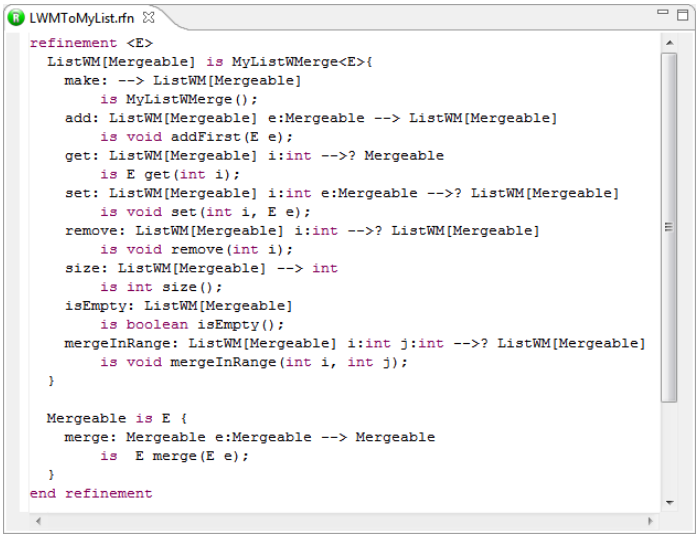


Figura3. Um mapa de refinamento de LWM para {`MyListWMerge<E>`, `Mergeable<E>`}.

nece uma implementação de listas mutáveis e, portanto, as operações que produzem valores do género ListWMerge[Mergeable] são todas enviadas para métodos **`void`**.

O refinamento apresentado na figura 3 também estabelece a correspondência entre o género Mergeable e a variável de tipo `E` e define que a operação merge corresponde a um método com assinatura `E merge(E e)`.

Uma vez definido o refinamento, o CONGU instrumenta `MyListWMerge.class` de forma a que, durante a execução de qualquer programa que use a classe `MyListWMerge`, a correcção do comportamento da classe seja verificada. Adicionalmente, o comportamento das classes que no programa instanciam `MyListWMerge<E>` é também verificado face ao especificado em Mergeable. Por exemplo, se o nosso programa inclui a classe `Color` que implementa `Mergeable<Color>` e, uma outra classe que cria e manipula objectos do tipo `MyListWMerge<Color>`, o comportamento de `Color` é verificado face ao especificado em Mergeable.

3 Correcção de Programas face a Especificações

Nesta secção apresentamos a noção de correcção de programas que é considerada no CONGU2 e discutimos os aspectos mais importantes da abordagem CONGU à monitorização desta correcção.

3.1 Propriedades de objectos induzidas por especificações

A correcção de um programa Java face a um módulo é definida tendo por base um mapa de refinamento estabelecendo uma ligação entre os dois lados. Na secção anterior foram mencionadas algumas das condições que a ligação de operações e predicados a métodos e construtores Java tem de obedecer. A ligação entre especificações e classes Java está sujeita a outras condições, capturadas na definição de mapa de refinamento.

Um *mapa de refinamento* consiste num conjunto V (variáveis de tipo) equipadas com uma pre-ordem $<$ e uma função $\mathcal{R}$ que envia: (1) cada especificação simples e *core* numa classe não genérica; (2) cada especificação paramétrica e *core* numa classe genérica com a mesma aridade; (3) cada especificação *core* que define um género $s < s'$, para uma subclasse de $\mathcal{R}(S')$, onde S' é a especificação que define s'; (4) cada especificação parâmetro numa variável de tipo V; (5) cada operação de uma especificação *core* num método da correspondente classe; (6) cada operação de uma especificação parâmetro numa assinatura de um método. Adicionalmente, (7) se uma especificação parâmetro S' define um sub-género do género definido na especificação S, então $\mathcal{R}(S') < \mathcal{R}(S)$; (8) se S é uma especificação paramétrica com parâmetro S', então tem de ser possível assegurar que qualquer tipo C que possa ser usado para instanciar $\mathcal{R}(S)$ tem métodos com as assinaturas definidas por $\mathcal{R}$ para a variável de tipo $\mathcal{R}(S')$ depois de trocar todas as ocorrências de $\mathcal{R}(S')$ por C.

Note-se que, no nosso exemplo, a condição (8) verifica-se porque `MyListWMerge` impõe que `E extends Mergeable<E>`. Uma vez que o interface `Mergeable` declara o método `E merge(E e)`, `E` só pode ser instanciado com classes `C` que implementem `Mergeable<C>` e, portanto, está assegurado que `C` tem o método `C merge(C e)`.

Seja $\mathcal{R}$ um mapa de refinamento entre um módulo $\mathcal{M}$ e um programa Java $\mathcal{J}$. Para $\mathcal{J}$ estar correcto face a $\mathcal{M}$, (i) as propriedades especificadas em $\mathcal{M}$ e (ii) as propriedades algébricas da noção de igualdade têm de ser verdadeiras em todas as possíveis execuções de $\mathcal{J}$.

As propriedades de uma especificação *core* S restringem o comportamento dos objectos do tipo $T_S = \mathcal{R}(S)$, enquanto que as propriedades de uma especificação S que é parâmetro, por exemplo $S'[S]$, restringem o comportamento dos objectos dos tipos T_S em $\mathcal{J}$ que sejam usadas para instanciar a variável de tipo $\mathcal{R}(S')$. As restrições impostas por axiomas e condições de domínio são diferentes:

Axiomas. Cada axioma de uma especificaçãoS define uma propriedadespara os objectos do tipo T_S que tem de ser verdadeira em todos os estados do objecto que sejam visíveis para o cliente.
Considere-se, por exemplo, o primeiro axioma do get em ListWM[Mergeable]. Se `lwm` é um objecto do tipo `MyListWMerge<C>`, está sujeito à seguinte propriedade: para todo o `e` de tipo `C`, se `e!=null`, então após a execução de `lwm.addFirst(e)`, `lwm.get(0).equals(e)` é verdadeira.

Domínios. Cada condição de domínio ϕ de uma operação op de uma especificação S define que, para todo o objecto do tipo T_S, sempre que ϕ é verdadeiro, a invocação de $\mathcal{R}(op)$ tem de retornar normalmente, sem levantar qualquer excepção.

As propriedade de objectos induzidas pelos axiomas e domínios que foram apresentadas definem uma noção de correcção. O CONGU usa, por omissão, uma noção mais forte que também impõe restrições aos clientes da classe T_S, quando invocam o método $\mathcal{R}(op)$. É exigido a estas classes que não passem o valor `null` como argumento a $\mathcal{R}(op)$ e que só invoquem o método quando a sua condição de domínio é verdadeira.

3.2 Monitorização das propriedades dos objectos

A estratégia usada no CONGU para monitorizar a correcção de um programa consiste em verificar os invariantes induzidos pelos axiomas no final de métodos específicos, de-

terminados pela estrutura dos axiomas. No caso dos axiomas em que a operação/predicado tem como primeiro argumento uma operação construtora, o invariante é verificado no final do método que refina a referida operação construtora. Por exemplo, o invariante induzido pelo primeiro axioma de get é verificado no final de **void** `addFirst(E e)` através da execução do seguinte código, onde `eOld` é uma cópia de `e` obtida à entrada do método.

```
if (eOld != null) {
  E e2 = this.clone().get(0);
  assert(e2 != null && e2.equals(eOld));
}
```

Note-se que todas as invocações que são realizadas de forma a verificar uma propriedade são realizadas sobre clones, se possível. De outra forma, os efeitos colaterais destes métodos afectariam os objectos monitorizados. Se o clone não é suportado, assume-se que os objectos da classe são imutáveis. De forma semelhante, o segundo axioma de get é verificado pelo seguinte código:

```
if (eOld ! = null)
  for (int i: rangeOfInt)
    if (i>0 && i<this.clone().size()) {
      E e2 = this.clone().get(i);
      assert((i-1)>=0 && (i-1)<thisOld.clone().size());
      E e3 = thisOld.clone().get(i-1);
      assert(e2!=null && e3!=null && e2.equals(e3));
    }
```

onde `rangeOfInt`, de tipo `Collection<Integer>`, é preenchido com os inteiros que são usados como argumento ou valores de retorno de algum dos métodos da classe. Apesar de neste caso, por se tratar de um tipo de dados primitivo, ser fácil arranjar outras formas de obter os valores do domínio alvo da quantificação universal, isto não acontece em geral. A estratégia adoptada, que passa por coleccionar os elementos do domínio que atravessam a fronteira da classe, é uma forma relativamente simples e leve de ter uma população representativa do domínio em causa.

Por outro lado, as propriedades induzidas por axiomas que têm uma variável como primeiro argumento da operação são verificadas no final do método que refina essa operação. Por exemplo, o último axioma de ListWM, que descreve uma propriedade de isEmpty, é verificado no final do método **boolean** `isEmpty()` pelo seguinte código, onde `result` é o valor de retorno de **boolean** `isEmpty()`.

```
if (result) assert(thisOld.clone().size()==0);
```

A igualdade entre inteiros é convertida em comparações com == enquanto que a igualdade de um género não primitivo, s, é convertida na invocação do método `equals` da classe T_S. É assim essencial que todas as classes envolvidas definam uma implementação apropriada deste método que, em particular, deve considerar dois objectos iguais apenas se têm comportamentos equivalentes quando se consideram os métodos que refinam alguma das operações de s (i.e., são *equivalentes do ponto de vista do seu comportamento*).

A correcção do `equals` é monitorizada no final deste método. Por exemplo, a monitorização de **boolean** `equals(Object other)` em `MyListWMerge` envolve:

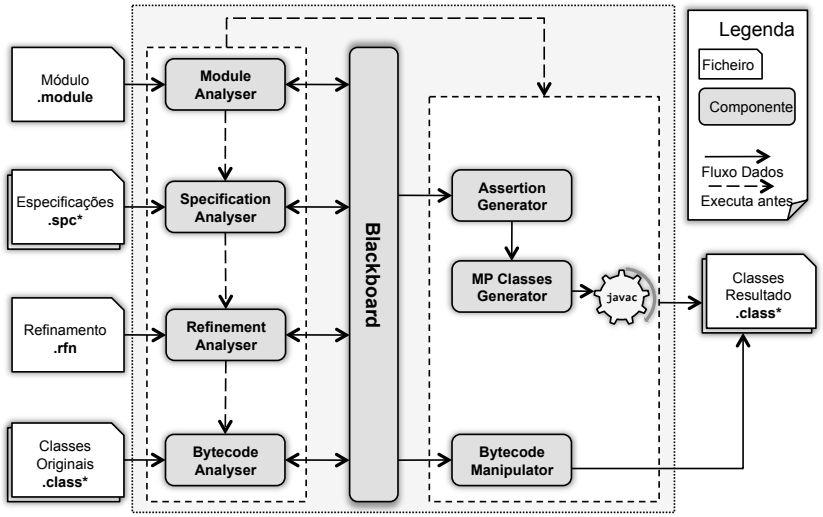


Figura4. A arquitectura do CONGU2.

```
if (result)
  for (int i: rangeOfInt)
    if (i>=0 && i<thisOld.clone().size()
             && i<otherOld.clone().size()) {
      E e1 = thisOld.clone().get(i);
      E e2 = otherOld.clone().get(i);
      assert(e1!=null && e2!=null && e1.equals(e2));
    }
```

Finalmente, a verificação de que as classes clientes não invocam um método que refina uma operação quando a condição de domínio é falsa, nem passam o valor **null** como argumento, pode ser facilmente realizada no início do método. Por exemplo, no caso do método **void** `set(`**int** `i,E e)`, isto é verificado por:

```
assert(e != null && i >= 0 && i < this.clone().size());
```

4 CONGU2

A versão 2 da ferramenta CONGU implementa a monitorização da correcção de programas Java descrita na secção anterior. Como mostrado na Figura 4, a ferramenta recebe um módulo, especificações, um mapa de refinamento e os binários de um programa Java. O programa é então transformado de forma a que, quando é executado com o CONGU, seja monitorizada a sua correcção. Isto é alcançado através da intercepção de todas as chamadas a métodos que refinam alguma operação/predicado, e redireccionado-as para classes monitoras de propriedades.

A ferramenta executa duas tarefas principais: a análise das diferentes fontes de *input* e a geração das classes de *output*. Na tarefa de análise, os maiores desafios colocados pela extensão da abordagem a especificações paramétricas e classes genéricas surge na análise dos refinamentos. Em 4.1 discute-se como foram endereçados estes desafios.

Apresentam-se depois os aspectos chaves da tarefa de geração: em 4.2, a instrumentação dos binários e, em 4.3, a geração das classes monitoras das propriedades.

4.1 Análise do mapa de refinamentos

A extensão do CONGU levanta vários desafios ao nível da análise dos refinamentos já que a verificação de várias condições a que os refinamentos têm de obedecer exige investigar os binários das classes referidas. Isto é alcançado recorrendo às capacidades de reflexão oferecidas pelo pacote `java.lang.reflection` do *API* do Java.

O processo de análise dos refinamentos tem duas fases: a primeira focada nas classes que refinam os géneros das especificações *core* e a segunda na verificação de condições relacionados com os géneros das especificações parâmetro.

A verificação de que um tipo não genérico tem os métodos mencionados no refinamento é bastante simples, já que basta obter o método especificado e depois verificar que o seu tipo de retorno é o esperado. A situação complica-se no caso de tipos genéricos por causa do mecanismo conhecido como *type erasure*, o qual torna um tipo genérico num *raw type* (substituindo todas as ocorrências das variáveis de tipo pelos seus limites superiores [8]). Por esta razão, é preciso uma estratégia mais sofisticada para verificar que uma classe genérica tem os métodos mencionados num refinamento: primeiro é preciso obter todos os métodos da classe e, para cada um destes, obter os seus tipos de parâmetros e o seu tipo de retorno genéricos e, depois, é preciso verificar se algum destes métodos é o esperado (um processo que tem de ser recursivo na estrutura dos tipos).

Na segunda fase da análise dos refinamentos são endereçadas as condições relacionadas com os géneros das especificações parâmetro. Recorde-se que, neste caso, os géneros não são refinados em tipos concretos e o que é preciso é assegurar que as classes que podem instanciar a correspondente variável de tipo têm os métodos necessários. Por exemplo, no nosso caso, nesta fase é verificado que toda a classe `C` que pode instanciar `E` em `MyListWMerge<E>` tem um método com a assinatura `C merge(C e)`. Isto é conseguido recorrendo aos limites superiores de `E` em `MyListWMerge` (no nosso caso há só um, mas em geral podem ser vários). Os métodos cuja assinatura envolve o `E` só precisam de ser pesquisados nos limites que também dependem de `E` (no nosso caso, o método `E merge(E e)` é procurado em `Mergeable<E>`).

4.2 Instrumentação dos binários

A monitorização dos programas Java assenta na intercepção das chamadas aos métodos relevantes por classes clientes. No CONGU2, esta intercepção é realizada através da instrumentação dos binários. As chamadas interceptadas são traduzidas em chamadas de um método correspondente numa classe monitora de propriedades, gerada pela ferramenta. Os principais desafios que este processo coloca são os seguintes:

Chamadas Internas. Como interceptar apenas as chamadas externas? As chamadas internas não podem ser monitorizadas, caso contrário o programa não termina.

Chamadas de super-classes. As chamadas de dentro de super-classes também não podem ser interceptadas.

Construtores. Como interceptar e redireccionar as chamadas aos construtores?

Clone e equals. O que fazer quando estes métodos não são redefinidos na classe?

A estratégia adoptada foi renomear todo o método *m* que precisa de ser interceptado (passa a chamar-se *m_Original*) e substituí-lo por um método com o mesmo interface, que redirecciona a chamada para `ClassPMonitoring.m(`**`this`**`, ...)`. Por outro lado, as chamadas internas são trocadas por chamadas ao método na sua forma renomeada. De forma semelhante, as chamadas a este método nas super-classes são também trocadas por chamadas ao método renomeado, o qual também é acrescentado nestas classes. Relativamente aos construtores, a sua intercepção é realizada de uma forma idêntica à dos métodos, apenas com a salvaguarda de que os construtores exigem chamadas de inicialização, as quais são removidas do método renomeado e inseridas no método que o substitui. O mesmo é feito para o `equals` e `clone`. Se a classe não redefine o `equals`, então é primeiro criado este método, o qual delega a chamada na super-classe. Se a classe não anuncia implementar `Cloneable` ou não redefine o método como público, então é gerado um método `clone_Original` que simplesmente retorna **`this`** (recorde-se que neste caso se assume que os objectos da classe são imutáveis). O método gerado é para exclusivo uso do processo de monitorização.

A implementação desta estratégia para instrumentação dos binários recorre à biblioteca de manipulação de bytecode ASM [4]. O ASM é uma biblioteca leve e eficiente, que, disponibilizando um API simples e bem documentado, suporta completamente o Java 6 e é distribuído sob uma licença *open-source* que possibilita a conveniente inclusão no pacote da ferramenta CONGU2.

4.3 Geração das Classes Monitoras de Propriedades

A monitorização das propriedades de objectos descrita na sub-secção 3.2 é executada por classes geradas pela ferramenta, a que chamámos *classes-MP*. Para cada classe C cujo comportamento precisa de ser monitorizado é gerada uma classe-MP, cujo nome resulta de juntar o sufixo `PMonitoring` ao nome de C.

Cada método que é interceptado tem um equivalente na respectiva classe-MP, na forma de um método de classe com o mesmo nome e tipo de retorno e cujos argumentos são os do método original mais um argumento *callee* com o tipo C (uma referência do objecto alvo da invocação original) e outro *flag* com o tipo Booleano (indicando se deve ser realizada a monitorização de propriedades).

A estrutura do corpo destes métodos segue o seguinte padrão: (1) guardar à entrada do método os valores dos vários argumentos; (2) verificar que a condição de domínio é verdadeira (apenas no caso da noção de correcção forte) e os argumentos não são **`null`**; (3) chamar o método original sobre o *callee* e guardar o valor de retorno, (4) verificar as propriedades requeridas e (5) retornar o valor de retorno original. Os passos (2) e (4), que são apenas executados se a *flag* é verdadeira, usam dois métodos de classe auxiliares, respectivamente, *mPre* e *mPos*. Estes testam as propriedades do objecto $callee$ de acordo com o descrito em 3.2 mas, em vez de chamarem os métodos originais, chamam os equivalentes na respectiva MP-classe, com a *flag* a falso (ver figura 5).

No passo (3), além de ser invocado o método original, é verificado que se a condição de domínio é verdadeira, então o método retorna normalmente. A chamada ao método original é envolvida por um *try-catch*, de forma a poder assinalar uma violação

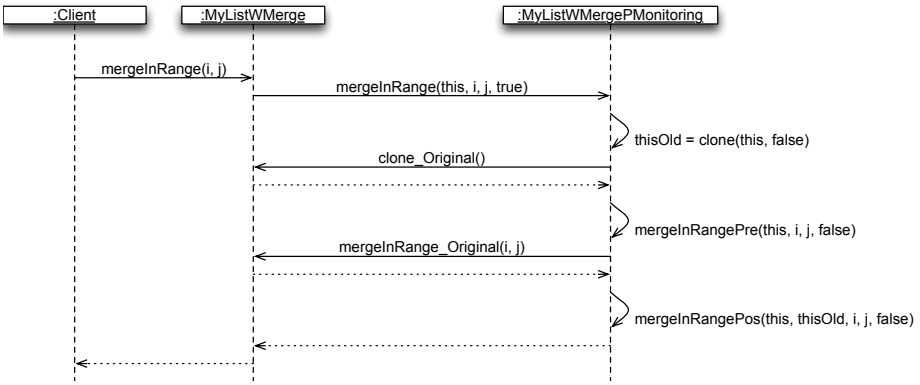


Figura5. Processo desencadeado por uma chamada externa a `mergeInRange(int,int)`.

da correcção do programa sempre que a condição do domínio é verdadeira e o método original levantar uma excepção. Se, pelo contrário, a condição de domínio for falsa, qualquer excepção apanhada é relançada.

Tudo o que foi até aqui descrito aplica-se a classes que refinam géneros *core* ou são usadas para instanciar uma classe genérica que refina um género *core*. No entanto, a monitorização das propriedades das especificações parâmetro exige um nível adicional de indirecção. Seja C uma classe genérica com parâmetro E que refina uma especificação $S[S']$. Apesar de uma classe usada para instanciar E em C ter uma correspondente classe-MP, o código gerado para monitorizar as propriedades de S que envolve invocar um método de E, não se pode comprometer com uma classe-MP específica. A solução adoptada foi gerar uma classe *dispatcher* para cada variável de tipo do mapa de refinamento. Esta classe, com os mesmos métodos que uma classe-MP, apenas serve para resolver a que classe-MP deve ser entregue cada chamada de um método, baseando-se para isso no tipo concreto do objecto *callee*. Na classe-MP de C, sempre que a verificação de uma propriedade exigir invocar um método de E, a chamada é feita sobre o correspodente *dispatcher*.

5 Conclusões

A importância de ferramentas que suportam a verificação da correcção de implementações face a especificações formais tem sido largamente reconhecida. Na última década, várias abordagens foram desenvolvidas que permitem monitorizar a fiabilidade de implementações em linguagens OO (ex., [1,3,5,6,9,11]). No entanto, e apesar da popularidade dos genéricos nestas linguagens, as abordagens existentes ainda não os suportam. Esta era também uma limitação do CONGU que foi superado com o CONGU2.

Neste artigo mostramos como a abordagem e a ferramenta CONGU foram estendidas de forma a apoiar a especificação de tipos de dados genéricos e sua implementação em termos de classes genéricas. A extensão da linguagem de especificação de forma a permitir a descrição de tipos de dados genéricos foi relativamente simples. Dado que o CONGU depende especificações baseadas em propriedades, essencialmente foram adoptadas especificações paramétricas semelhantes às disponíveis em várias linguagens

de especificação algébricas. A fim de colmatar o fosso entre especificações paramétricas e classes genéricas propusemos uma nova noção de mapa de refinamento em torno da qual foi definida uma nova noção de correcção de programas face a especificações. Tanto quanto sabemos, este aspecto não foi endereçado noutros contextos. Existem outras abordagens que lidam com especificações arquitectónicas envolvendo especificações paramétricas, como por exemplo [2], mas têm como alvo programas ML. Por outro lado, as relações entre especificações algébricas e programas OO de que temos conhecimento consideram exclusivamente especificações simples e sem estrutura.

O CONGU2 assegura a monitorização desta noção de correcção mais abrangente, que, no caso dos tipos de dados genéricos, envolve verificar que tanto a classe que implementa o tipo de dados como as classes usadas para instanciá-lo estão em conformidade com o que foi especificado. Com o CONGU2, a verificação da correcção de um programa em tempo de execução passa a ser aplicável a um conjunto de situações em que o suporte automático para a detecção de erros é ainda relevante: os genéricos são reconhecidamente complicados de dominar e, portanto, a correcção de implementações com genéricos mais difícil de atingir.

Referências

1. S. Antoy and R. Hamlet. Automatically checking an implementation against its formal specification. *IEEE Transactions on Software Engineering*, 26(1):55–69, 2000.
2. D. Aspinall and D. Sannella. From specifications to code in CASL. In *Proc. Algebraic Methodology and Software Technology (AMAST) 2002*, volume 2422 of *LNCS*, pages 1–14. Springer, 2002.
3. M. Barnett and W. Schulte. Runtime verification of .NET contracts. *Journal of Systems and Software*, 65(3):199–208, 2003.
4. E. Bruneton, R. Lenglet, and T. Coupaye. ASM: A code manipulation tool to implement adaptable systems. In *Proc. ACM SIGOPS France Journées Composants 2002: Systemes a composants adaptables et extensibles (Adaptable and extensible component systems)*, 2002.
5. F. Chen and G. Rosu. Java-MOP: A monitoring oriented programming environment for Java. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2005*, volume 3440 of *LNCS*, pages 546–550, 2005.
6. Y. Cheon and G.T. Leavens. A runtime assertion checker for the Java Modeling Language (JML). In *Proc. International Conference on Software Engineering Research and Practice (SERP 2002)*, pages 322–328, 2002.
7. Contract Based System Development. http://gloss.di.fc.ul.pt/congu/.
8. J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification, Third Edition*. Prentice Hall, 06 2005.
9. J. Henkel and A. Diwan. Discovering algebraic specifications from Java classes. In *Proc. ECOOP 2003*, volume 2743 of *LNCS*, pages 431–456, 2003.
10. G.T. Leavens, Y. Cheon, C. Clifton, C. Ruby, and D.R. Cok. How the design of JML accommodates both runtime assertion checking and formal verification. *Science of Computer Programming*, 55(1–3):185–208, 2005.
11. B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall PTR, 2nd edition, 1997.
12. I. Nunes, A. Lopes, and V. Vasconcelos. Bridging the gap between algebraic specification and object-oriented generic programming. In *Runtime Verification*, pages 115–131, 2009.
13. I. Nunes, A. Lopes, V. Vasconcelos, J. Abreu, and L.S. Reis. Checking the conformance of Java classes against algebraic specifications. In *ICFEM'06*, volume 4260 of *LNCS*, pages 494–513. Springer, 2006.

Separation of Concerns in Parallel Applications with Class Refinement

Matheus Almeida and João Luís Sobral

Departamento de Informática
Universidade do Minho,
Braga, Portugal

Abstract. Parallel programming is becoming increasingly important since the popularization of multi-core processors. Traditional programming techniques that take advantage of these processors lack structure in the sense that the parallelization artefacts are mixed with the base code. This leads to problems in reusing, debugging and maintaining both the base code and the parallelization code. This paper presents and compares a new approach to separate those concerns. This approach is based on the concept of Object-Oriented Programming inheritance and it is called Class Refinement. Since the concepts and abstractions are similar to those on Object-Oriented, the learning curve is much smaller than using, for instance, the Aspect Oriented (AOP) approach.
We show that the performance overhead of using Class Refinement is close to the AOP approach and minimal compared to the traditional programming style.

1 Introduction

The solution adopted to overcome the problems of the increase of frequency in processors [1] is to integrate into a single CPU a set of independent processing units (*cores*). With this approach, processor designers no longer need to raise clock frequencies to increase computational power. The trend is the continue increase of the number of cores.

The older variant of parallel computing but still very important today is related to distributed computing (eg.: Cluster) where the computation is performed across a number of nodes connected by a network. The most important benefits of this approach are the large number of nodes that can be interconnected and the fact that each node can be composed by commodity hardware making it a low cost solution.

Both multi-core and cluster computing require a different programming style from sequential programming, as programmers need to specify parallel activities within applications. Thus, the development of parallel applications requires knowledge of traditional programming and expertise in parallel execution concerns (eg.: data partition, thread/process communication and synchronization). Generally, these two concerns are mixed because the code that supports the parallel execution is *injected* into the core functionality (coded sequentially),

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 67–78

resulting in *tangled code*. The lack of structure of this approach also leads to scattered code since the code to enable parallel execution is spread over different *classes/modules* of the *base/domain* code. The main drawbacks of that approach are mostly noticed in the greater effort that is necessary to understand both the parallel structure of the program and the base algorithm and in the difficulty to reuse or debug functionalities.

Previous studies [2,3] argue the separation of the core functionality from the parallelization structure which allows :

1. better maintenance and reuse of the core functionality, reducing or eliminating the problem of *code tangling* and *scattering*;
2. easier understanding of the parallel structure and better reuse of the parallel code;
3. enhancement of the parallelization structure by promoting incremental development.

Aspect Oriented Programming [4] aims to separate and modularize crosscutting concerns that are not well captured by the Object-Oriented (OO) approach. It was already used successfully to separate the parallelization structure from the base/domain code [2, 3, 5]. The experience gained led us to investigate the use of Class Refinement to achieve a similar goal. The main purpose is to ease the migration of programmers since the rules and abstractions are similar to the ones found in Object-Oriented programming (OOP).

The remainder of this document is structured as follows. Section 2 gives an overview and a comparison of the techniques studied in this paper to separate concerns in parallel applications. The next section shows the implementation of a case study and the overhead caused by the separation of concerns. The conclusion and the future work are presented in Section 4.

2 Tangled, AOP and Class Refinements

This section introduces the problems with the traditional approach for the parallelization of applications and compares two other approaches that allow separation of concerns. AOP and Class Refinement allow the separation of the parallelization into well defined modules promoting modularization [6–8]. Both approaches need that the base code exposes *entry points* where additional code can be appended. In other words, some functionalities in the base code have to be separated into methods to support the attachment of code in the new units of modularity. When those *entry points* are not available, *code refactoring* is needed.

2.1 Traditional Approach

Traditional techniques to parallelize applications are invasive. Program 1 illustrates the problem of invasive modification by showing the simplified cluster

oriented parallelization of a molecular dynamics simulation [9] that will be detailed in section 3. In black it can be seen the base code and in red (italic) the parallelization statements.

Once the domain code is populated with artefacts regarding the parallelization concerns, modularity is lost. This doesn't allow, for instance, to change the parallelization to match other target platforms (eg.: Shared Memory) or to perform incremental development to enhance the parallelization or the domain code. Both codes are glued and dependent on each other.

```
public class MD {
   Particle [] one; // Vector with all particles
   int mdsize; // Problem size (number of particles)
   int movemx; // Number of interactions

    //Declare auxiliary variables to MPI parallelization
    double [] tmp_xforce;
    double [] tmp_yforce;
    double [] tmp_zforce;
    ...
    public void runiters throws MPIException {

       for (move = 0; move < movemx; move++) { // Main loop
          for (i = 0; i < mdsize; i++) {
             one[i].domove(side); // move the particles and
          }                       // update velocities
    ...
    MPI.COMM_WORLD.Barrier();
    computeForces(MPI.COMM_WORLD.Rank(),MPI.COMM_WORLD.Size());
    MPI.COMM_WORLD.Barrier();
    for (i = 0; i < mdsize; i++) {  //Copy forces to temp vector
       tmp_xforce[i] = one[i].xforce;  // to use in MPI operation
       tmp_yforce[i] = one[i].yforce;
       tmp_zforce[i] = one[i].zforce;
     }
     //Global reduction
     MPI.Allreduce(tmp_xforce,0,tmp_xforce,0,mdsize,MPI.DOUBLE,MPI.SUM);
     MPI.Allreduce(tmp_yforce,0,tmp_yforce,0,mdsize,MPI.DOUBLE,MPI.SUM);
     MPI.Allreduce(tmp_zforce,0,tmp_zforce,0,mdsize,MPI.DOUBLE,MPI.SUM);

     //Update forces based in reducted values
     //Scale forces and calculate velocity
```

Program 1: MD cluster based parallelization.

2.2 AOP Technique

Aspect Oriented Programming [4] aims to separate and modularize crosscutting concerns that are not well captured by the Object-Oriented (OO) approach. Aspects are units of modularization that encapsulate code that otherwise would be scattered and tangled with the base code.

Crosscutting concerns can be either static or dynamic. Static crosscutting allows the redefinition of the static structure of a type hierarchy. For instance, fields of a class can be added or an arbitrary interface can be implemented. For dynamic crosscutting, AOP introduces the concepts of *join point* and *advice*. Join point is a well defined place in the base code where arbitrary behaviour can be attached and advice is the piece of code to execute when a join point is reached. A set of join points can be defined by the use of the pointcut designator that allows the use of logical operators to define an arbitrary point in the program flow.

In program 2 an Aspect example is shown. This aspect traces all calls to the method *Deposit* defined in *Bank* class that have one argument of type *int* (line 3). Before that method being called (line 5), a piece of *advice* is executed. Lines 6 and 7 correspond to the advice.

Weaving is the mechanism responsible to compile the aspects. It merges both the aspects and the base classes. This process can be done either in the compilation phase or during class loading.

In the remainder of this paper whenever we'll use the term AOP we will be referring to the most mature and complete AOP implementation, AspectJ [10].

```
1  public aspect Logging {
2      int call = 0;
3      pointcut deposit() : call (void Bank.Deposit(int));
4
5      before() : deposit() {
6         Logger.log(...);
7         call ++;
8     }
9 }
```

Program 2: AOP logging example. AspectJ syntax.

2.3 Class Refinement Technique

Object-Oriented languages allow the extension of classes by means of inheritance. The new class (subclass) inherits a subset or the complete set of the superclass state and behaviour. The subclass has the ability to override that behaviour or introduce a new one. This mechanism can be seen as a layer where additional,

more specific behaviour can be attached. Thus, the fact that the subclass needs to be instantiated does not solve entirely the problem of separation of concerns. The access of the new behaviour or state defined in the subclass has to be done explicitly by using, for instance, the name of the subclass. Clearly, this solution does not scale if we want to encapsulate others parallelization mechanisms, each in its own module (eg.: subclass) because it is required to make changes to client modules to compose them.

Batory [11] proposed that refinement "*is a functionality addition to a program, which introduces a conceptually new service, capability, or feature, and may affect multiple implementation entities*". Others definitions are more restrictive and thus specific to a particular area[1].

In this study, we define Class Refinement as the ability to extend a class by means of inheritance but instead of creating a new scope (subclass), the refined class takes the name of the original class. In terms of implementation, the original class is rewritten to include the modifications defined in the refinements. Composition order becomes important since refinements are class rewritings (note: the composition order is also important in traditional OO inheritance, although it is implicitly specified by the inheritance chain and method lookup).

```
1  public class Logging refines Bank {
2     int call = 0;
3     @Override
4     public void Deposit(int value){
5         Logger.log(...);
6         call ++;
7         super.Deposit(value);
8      }
9  }
```

Program 3: Class Refinement example.

In program 3 the same example is given but using the Class Refinement approach. The similarities with Object-Oriented inheritance are huge and besides the word *refines* in Line 1, this piece of code could belong to a Bank's subclass. The main difference is that the class *Logging* does not need to be explicitly instantiated because it will rewrite the class Bank. This means that calls to the *Deposit* method in a Bank object will trigger the execution of lines 5 and 6 when the refinement is applied to the base code.

For the rest of this paper, we'll present Class Refinements implemented with GluonJ [12] (although, we do not strictly follow GluonJ's syntax). GluonJ supports refinements by means of inheritance using Java annotations. This allows the use of a standard Java compiler (eg.: javac) to compile both the base code

[1] One example belongs to formal methods (eg.: Refinement Calculus)

and the refinements. When more than one refinement is applied, it is necessary to define an order of composition because different refinements can be applied to the same class (eg.: two refinements can override the same method). Refinements are applied in load-time by the GluonJ mechanism [13] using the order defined in a specific container called *Glue class*.

2.4 Comparison

In this section we present a comparison among the approaches previously discussed to separate concerns against the traditional (tangled) parallel programming approach. We compare each approach using a set of properties that we think are fundamental to a major acceptance by the community and to build better and modular applications.

	ClassRefinement	*AOP(AspectJ)*	*Traditional*
$OO-Like$	Yes	No	Yes
$Modularity$	$Good$	$Good$	$Poor$
$Context$	$VeryGood$	$VeryGood$	$Excellent$
$Composition$	$Good$	$Average$	$Poor$
$Reusability$	$Average$	$VeryGood$	$Poor$
$Usability$	$VeryGood$	$Hard$	$Excellent$
$Performance$	$VeryGood$	$VeryGood$	$Excellent$
$UnanticipatedEvolutions$	Yes	Yes	No

Table 1: Comparison of Approaches.

Being *OO-Like* is important for a major acceptance from the community. The AOP approach is the weaker in this case because programmers need to learn new concepts different from Object-Oriented Programming and it also requires an Aspect compiler (eg.: ajc). Class Refinement, on the other hand, shares the same concepts with the Object-Oriented approach and it can be compiled with a standard compiler like *javac*.

Class Refinement and AOP allow improvements in modularity since new concerns are localized in new units of modularity (Refinement and Aspect). Traditional parallel programming techniques present poor modularity due to the mentioned code tangling.

Context information is the ability to access behaviour or information defined in the base code. Since method overriding is the finest grain to change the base class, Class Refinements can access almost everything except local information in methods. For instance, method overriding doest not allow to reuse parts of the overridden method. The same situation happens with AOP but with the difference that some context information can be retrieved using reflection (eg.: *thisJoinPoint*) bringing performance penalties. On the other hand, the tangled

approach, where code can be inserted anywhere, has excellent context information access.

In terms of composition, the order in which the refinements and the aspects are applied plays an important rule and can be tricky. Nevertheless, both approaches are superior to the tangled version, as it is not possible to compose code that is not made in a modular manner. Composition using Class Refinements is better than with AOP since we explicitly specify which refinements must be applied to the base code. For instance, in GluonJ, a specific *Glue class* specifies the set and order of refinements. AOP lacks such kind of explicit composition step and clear composition rules.

Re-usability is a topic still in research in the case of Class Refinements. First implementations of reusable mechanisms using Class Refinement share the problem of explicitly defining the name of the class to refine making it an average solution. There are reusable implementations of concurrency mechanisms implemented using AOP [14] and the base code can be reused as well. In the tangled approach, the base code and the parallelization structure cannot be reused because they are intrinsically glued.

The tangled version is the easiest to use because it is the simplest approach. Class Refinement, since it is OO-like and based on inheritance, borrows most of its concepts in Object-Oriented Programming, making it easier to learn and use. AOP is the hardest because it introduces new concepts (eg.: join-point model, pointcuts) and it even changes some Object-Oriented properties (eg.: methods with body in Interfaces).

The change of the parallelization to match a new target platform can be seen as an unanticipated evolution. The tangled version is the weakest because its hard to reuse the base code. The same does not happen to the other two approaches as we have been seen.

In terms of performance, in section 3.2 we'll compare the approaches in more detail.

3 Case Study

We present the parallelization of a Molecular Dynamics (MD) algorithm that makes part of the Java Grande Forum [9] benchmark suite. Molecular Dynamics are important algorithms to simulate the interaction of microscopical particles (eg.: atoms) in a great variety of fields (eg.: Physics, Biology or Medicine).

Figure 1 shows the main steps of a generic MD algorithm. The first step is to assign particles its initial position. The algorithm then iterates until some condition is met (eg.: number of iterations). At each iteration, it is calculated the force on each particle due to the interaction with all others particles (main computational cost). The next step is to determine the new position of each particle and increment the time step.

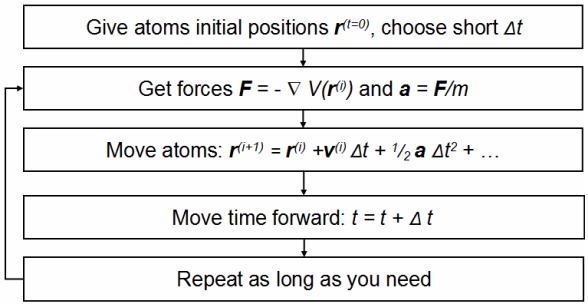


Fig. 1: Typical MD Algorithm [15]

3.1 Implementation

Figure 2 shows a simplified class diagram. The class **MD** contains all the information about the simulation including references to all particles. The method *runiters* implements the iterations of the simulation. The class **Particle** contains 9 variables representing the position, velocity and force for all coordinates in 3 dimensional space. The method force calculates the force for that specific particle with all others particles in the simulation.

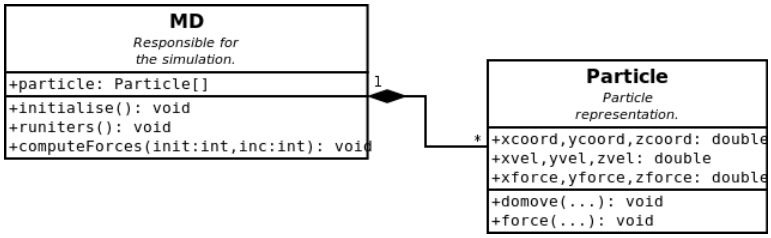


Fig. 2: Simplified MD class diagram.

To implement the shared memory parallelization using Class Refinement, the refinements listed in program 4 and 5 were created. The parallel algorithm implemented is based on the idea that the computation of the forces can be done in parallel, where each computation unit (Thread) calculates the forces for a subset of the total number of particles. When all of these computation units end, the result is merged (reduce operation).

The refinement of the MD class introduces a new data structure that will save temporary calculations before the reduce operation in the method *computeForces*.

The refinement of the class Particle is needed to use the new data structure created in the refinement *RefMD* that saves temporary computation of the forces.

```
public class RefMD refines MD {
  public static double[][] lforcex;
  ...//same structure to forcey and forcez

  @Override
  public void runiters(){
     //initialise new data structures
     //call original runiters to initialise data structures
     //of the original simulation
     super.runinters();
}

@Override
public void computeForces(...){
   //Spawn threads to compute forces in parallel
   //Join threads and reduce the values calculated in parallel

}
```

Program 4: MD refinement.

```
public class RefParticle refines Particle {

@Override
public void setForceX(...){
    //save in a new data structure created in RefMD
}

//Same to other components of the Force (y and z)
}
```

Program 5: Particle refinement.

To implement the distributed memory version, the Message Passing Interface (MPI) library was used to handle the creation, communication and synchronization of processes. Since the MPI parallelization is based in the Single Process Multiple Data (SPMD) methodology, only one refinement was needed and is presented in program 6.

The algorithm is the same as the shared memory version but instead of threads, the computational units are processes that have their own memory space and communicate through messages. The refinement *RefMPI* overrides the method *computeForces* to allow partitioning the computation. When each process ends, information is interchanged to continue the algorithm with updated values.

To take advantages of modern clusters that have hundreds or thousands of nodes where each node is composed by multi-core processors, a Hybrid version can be seen as the computation using Distributed and Shared memory parallelization. The creation of an Hybrid is just a matter of composing the refinements in the right order as shown in program 7. The first refinement being applied is *RefMD* and the last *RefMPI*. Thereafter, the behaviour defined in the refinement of the distributed memory version is the first being executed and then the behaviour in *RefMD*.

```
public class RefMPI refines MD {
 @Override
    public void computeForces(int init, int inc){
       super.computeForces(mpiRank,mpiSize);

       //Interchange information with MPI_AllReduce
       //Update state

       }
}
```

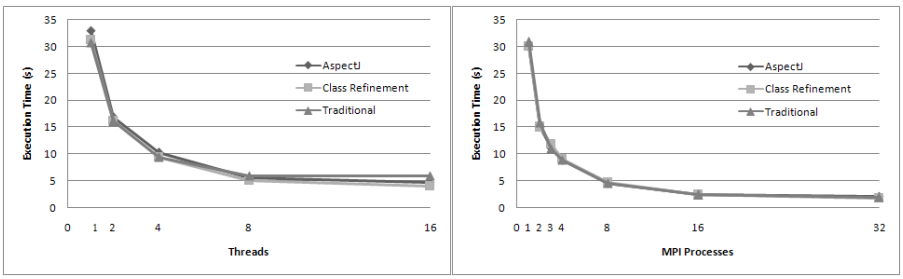
Program 6: MD refinement for MPI.

```
  applyRefinement
     RefMD,
     RefParticle,
     RefMPI
```

Program 7: Composition of Refinements.

3.2 Benchmarks

The benchmarks measure the execution time of three implementations of the algorithm explained in section 2 in both shared memory (Figure 3a) using multi-threads and distributed memory (Figure 3b) using MPI. As we expected because



(a) Shared memory (b) Distributed memory

Fig. 3: Benchmarks

the mechanism of class rewriting, the overhead caused by the Class Refinement mechanism is virtually zero and similar to the AOP approach. The difference for 8 threads can be explained by the use of concurrency mechanisms presented in *Java 1.6* that perform better for high number of threads compared to the implementation used in the JGF benchmark (*Java1.2*) (executors with thread pool). Similar results were obtained for the distributed memory parallelization. The differences in execution time are minimal in the 3 approaches.

4 Conclusion

This paper presented a new approach to solve the problem of separation of concerns in the parallelization of applications. It is based in Object-Oriented inheritance to ease the migration of programmers and to be compatible with standard compilers.

We presented a comparison among different approaches to identify the advantages and disadvantages of each methodology. There is no clear winner but the conclusions are important to understand what are the main important properties that must be presented in a system to allow a better and most complete separation of concerns. AOP and Class Refinement allow the creation of a new unit of modularity, thus allowing to deal with the inclusion of new concerns or with unanticipated changes in a modular way. Both of the approaches showed similar performance.

The main drawbacks of AOP are the need to learn new concepts and the fact that the compilation is done by a specific compiler. The Class Refinement approach is better in this case because it shares the same concepts with Object-Oriented inheritance and the compilation is done with a standard compiler.

The case study illustrated the use of Class Refinement and the benefits from its use compared to regular OO inheritance. The ability to compose the refinements and to choose what refinements must be applied in load-time is a great advantage compared to other approaches. The benchmarks showed that the overhead of using Class Refinement and AOP is little compared to traditional and invasive approaches.

Current work includes the implementation of larger case studies and optimized reusable mechanisms for parallel computing based on Class Refinement.

In the longer term, the creation of a new tool or the optimization of an existent one [12] that implements the concept of Class Refinement is an option.

5 Acknowledgements

This work was supported by the project Parallel Refinements for Irregular Applications (UTAustin/CA/0056/2008) funded by Portuguese FCT and European funds.

References

1. G. Koch, "Discovering multi-core : Extending the benefits of moore's law," *Technology@Intel Magazine*, 2005.
2. R. C. Gonçalves and a. L. Sobral, Jo "Pluggable parallelisation," in *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, (New York, NY, USA), pp. 11–20, ACM, 2009.
3. J. Sobral, "Incrementally developing parallel applications with aspectj," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pp. 10 pp.–, April 2006.

4. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *ECOOP'97 - Object-Oriented Programming - 11th European Conference*, vol. 1241, pp. 220–242, June 1997.
5. B. Harbulot and J. R. Gurd, "Using aspectj to separate concerns in parallel scientific java code," in *AOSD 2004 Conference*, 2004.
6. D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.
7. D. L. Parnas, "Designing software for ease of extension and contraction," *Software Engineering, IEEE Transactions on*, vol. SE-5, no. 2, pp. 128–138, 1979.
8. E. W. Dijkstra, *A Discipline of Programming*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1978.
9. L. A. Smith, J. M. Bull, and J. Obdrzálek, "A parallel java grande benchmark suite," in *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, (New York, NY, USA), pp. 8–8, ACM, 2001.
10. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold, "Getting started with aspectj," *Commun. ACM*, vol. 44, no. 10, pp. 59–65, 2001.
11. Y. Smaragdakis and D. Batory, "Mixin layers: An object-oriented implementation technique for refinements and collaboration-based designs," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 2, pp. 215–255, 2002.
12. M. Nishizawa and S. Chiba, "A small extension to java for class refinement," in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, (New York, NY, USA), pp. 160–165, ACM, 2008.
13. S. Chiba and M. Nishizawa, "An easy-to-use toolkit for efficient java bytecode translators," in *GPCE '03: Proceedings of the 2nd international conference on Generative programming and component engineering*, (New York, NY, USA), pp. 364–376, Springer-Verlag New York, Inc., 2003.
14. C. A. Cunha, J. a. L. Sobral, and M. P. Monteiro, "Reusable aspect-oriented implementations of concurrency patterns and mechanisms," in *AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development*, (New York, NY, USA), pp. 134–145, ACM, 2006.
15. K. Nordlund, "Md algorithm." "http://en.wikipedia.org/wiki/File:Mdalgorithm.PNG".

Uma Estrutura de Dados Métrica Genérica, Dinâmica, em Memória Secundária

Ângelo Sarmento e Margarida Mamede

CITI, Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829–516 Caparica, Portugal
amlsarmento@gmail.com, mm@di.fct.unl.pt
http://di.fct.unl.pt

Resumo Apresenta-se uma adaptação da estrutura de dados métrica RLC a memória secundária. A RLC é genérica, dinâmica e eficiente quando comparada com outras estruturas de dados métricas implementadas em memória central, onde apenas se contabiliza o número de distâncias calculadas. Neste trabalho, os testes experimentais comparam a RLC com três estruturas de dados implementadas em memória secundária, abrangem dicionários e conjuntos de imagens e medem, quer o número de distâncias calculadas, quer o número de operações de entrada e de saída efectuadas. Os resultados mostram que a RLC é muito eficiente em pesquisas por proximidade e muito competitiva em inserções.

Abstract We introduce an adaptation of the RLC metric data structure to secondary memory. RLC is generic, dynamic, and efficient when compared with other metric data structures implemented in main memory, where performance is analysed only in terms of the number of distance computations. In this work, the experimental study compares RLC with three data structures implemented in secondary memory, comprises two dictionaries and two sets of images, and evaluates both the number of distance computations and the number of I/O operations performed. The results show that RLC is very efficient for range queries and very competitive for insertions.

Keywords: Algorithms, data structures, metric spaces, range search.

1 Introdução

Em muitas áreas (como, por exemplo, biologia computacional, sistemas de informação geográfica ou multimédia), é necessário encontrar os objectos que mais se assemelham a um dado objecto. Essa semelhança é traduzida por uma função que calcula a *distância* entre dois objectos, com base nas suas características. Assume-se que, quanto menor for a distância, mais semelhantes são os objectos.

Devido aos formatos complexos dos dados (e.g. vídeos, imagens, sons, impressões digitais ou sequências de ADN) e à elevada quantidade de informação,

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 79–90

é crucial que não se calcule a distância entre cada objecto da base de dados e o objecto fornecido na pesquisa, sempre que uma procura é realizada. O objectivo das *estruturas de dados métricas* é minimizar o número de cálculos de distâncias entre objectos efectuados nas operações sobre a base de dados.

Não obstante nos últimos anos terem sido propostas muitas estruturas de dados métricas [10,17], a maioria é para ser implementada em memória central, sofrendo das limitações inerentes ao espaço disponível. Se nos cingirmos às estruturas de dados métricas implementadas em memória secundária, as alternativas existentes são classificadas como *genéricas*, quando suportam qualquer tipo de objectos e qualquer função de distância, ou como *não genéricas*, no caso contrário. As estruturas OP-tree [13] e SDI-tree [14] são exemplos do segundo grupo, aceitando apenas dados vectoriais. Todas as estruturas de dados métricas genéricas e implementadas em memória secundária são *dinâmicas*, ou seja, permitem realizar actualizações ao seu conteúdo após o carregamento inicial dos dados. No entanto, algumas, como a M-tree [2], a Slim-tree [15] e a DF-tree [16], só suportam a operação de inserção de um novo objecto. É possível efectuar inserções e remoções na SM-tree [12] e na D-Index [4].

Este trabalho aborda o problema da pesquisa por proximidade em estruturas de dados métricas genéricas, dinâmicas e implementadas em memória secundária, estudando uma adaptação da estrutura de dados RLC a memória secundária. A RLC, cujo nome por extenso é *Recursive Lists of Clusters*, é uma estrutura de dados métrica genérica, dinâmica (suportando inserções e remoções) e implementada em memória central. Foi proposta em [6], mas a sua definição inicial foi simplificada (para depender de menos um parâmetro) e os novos algoritmos foram analisados em [7], tendo-se provado que o número médio de distâncias calculadas no carregamento de uma base de dados com n objectos é $O(n \log n)$, numa inserção é $O(\log n)$ e numa remoção é $O(\log^2 n)$. Em todos os estudos comparativos efectuados, quer com dados gerados aleatoriamente [6,7], quer com dados reais (dicionários de línguas naturais [8], imagens de rostos [1] e excertos de música [3]), a RLC mostrou ter um óptimo desempenho.

A adaptação da RLC a memória secundária foi um desafio. Por um lado, as estruturas de dados em memória secundária são muito diferentes das implementadas em memória central. A RLC é uma excepção. Portanto, o bom desempenho comparativo da RLC, obtido em trabalhos anteriores, poderia não se manter. Por outro lado, como a avaliação das estruturas de dados métricas em memória secundária também depende do número de leituras e do número de escritas em ficheiro, para além do número de distâncias calculadas, era necessário alterar o desenho da implementação e os resultados poderiam não ser satisfatórios.

O artigo está organizado da seguinte forma. Na Secção 2, introduzem-se as definições básicas e, na Secção 3, descreve-se brevemente a RLC e a sua adaptação para memória secundária. Depois, na Secção 4, caracterizam-se os espaços métricos usados nos testes experimentais, que são baseados em dicionários e conjuntos de imagens. Na Secção 5, analisam-se os resultados experimentais, que comparam a RLC com três estruturas de dados. Finalmente, a Secção 6 contém alguns comentários ao trabalho desenvolvido e tópicos para trabalho futuro.

2 Definições Básicas

A noção de semelhança ou proximidade entre objectos baseia-se no conceito formal de espaço métrico.

Seja $(\mathcal{U}, d)$ um *espaço métrico*. Ou seja, $\mathcal{U}$ é o *universo* dos *pontos* ou *objectos* e $d : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ é uma função real, chamada *distância* ou *métrica*, que satisfaz as seguintes propriedades, para quaisquer $x, y, z \in \mathcal{U}$:

- (não negatividade) $d(x, y) \geq 0$;
- (identidade) $d(x, y) = 0 \Leftrightarrow x = y$;
- (simetria) $d(x, y) = d(y, x)$; e
- (desigualdade triangular) $d(x, y) \leq d(x, z) + d(z, y)$.

Uma *base de dados sobre* $(\mathcal{U}, d)$ é um conjunto finito $B \subseteq \mathcal{U}$.

Por exemplo, para qualquer $k \geq 1$, os pares $(\mathbb{R}^k, d_1)$ e $(\mathbb{R}^k, d_2)$ são espaços métricos, onde d_1 é a distância de *Manhattan* e d_2 é a distância *euclidiana*:

$$d_1((p_1, p_2, \ldots, p_k), (q_1, q_2, \ldots, q_k)) = \sum_{i=1}^{k} |p_i - q_i| \ , \tag{1}$$

$$d_2((p_1, p_2, \ldots, p_k), (q_1, q_2, \ldots, q_k)) = \sqrt{\sum_{i=1}^{k} (p_i - q_i)^2} \ . \tag{2}$$

Quando o universo é composto por palavras (i.e., sequências de caracteres), é frequente recorrer-se à distância de *Levenshtein*, que indica o número mínimo de operações de edição necessárias para transformar uma palavra na outra. Cada operação de edição pode ser a inserção de um carácter, a remoção de um carácter ou a substituição de um carácter por outro. Formalmente, sejam $X = x_1\, x_2\, \cdots\, x_m$ e $Y = y_1\, y_2\, \cdots\, y_n$ duas palavras (com $m, n \geq 1$). A distância entre X e Y é dada por $d_L(X, Y) = d_{X,Y}(m, n)$, sendo a segunda função definida recursivamente da seguinte forma, onde $\mathrm{dif}(a, b)$ é uma função que vale 0, quando $a = b$, e vale 1, no caso contrário.

$$d_{X,Y}(i, j) = \begin{cases} i, & \text{se } i \geq 0 \text{ e } j = 0; \\ j, & \text{se } i = 0 \text{ e } j > 0; \\ \min(\ d_{X,Y}(i-1, j-1) + \mathrm{dif}(x_i, y_j), & \\ \qquad 1 + d_{X,Y}(i, j-1), & \\ \qquad 1 + d_{X,Y}(i-1, j)\ ), & \text{se } i > 0 \text{ e } j > 0. \end{cases} \tag{3}$$

Dadas uma base de dados B, sobre um espaço métrico $(\mathcal{U}, d)$, e uma *interrogação* (q, r), onde $q \in \mathcal{U}$ e $r \geq 0$, o *problema da pesquisa por proximidade* consiste em encontrar todos os objectos de B cujas distâncias a q não excedem r, i.e., $\{x \in B \mid d(x, q) \leq r\}$. Chama-se a q o *ponto da interrogação* e a r o *raio da interrogação*.

O principal objectivo das estruturas de dados métricas [10,17] é minimizar o número de cálculos de distâncias entre objectos executados durante as pesquisas. Apesar das suas muitas diferenças, todas elas se baseiam na simetria e

na desigualdade triangular para incluir e excluir do conjunto resposta alguns objectos da base de dados, sem calcular as respectivas distâncias ao ponto da interrogação.

3 Recursive Lists of Clusters

Nesta secção define-se a RLC, apresentam-se resumidamente os algoritmos de inserção e de pesquisa (por proximidade) e descrevem-se os aspectos mais importantes da sua implementação em memória secundária.

3.1 Definição da RLC

Basicamente, a RLC (*Recursive Lists of Clusters*) guarda os objectos da base de dados numa lista (ou sequência) de agrupamentos. Cada *agrupamento* (*cluster*) possui:

- um *centro* c, que é um objecto da base de dados;
- um *raio* r, que é um número real não negativo; e
- um *interior* I, que contém um conjunto de objectos da base de dados cujas distâncias a c pertencem ao intervalo $]0, r]$. (Note que a exclusão de zero impede que o centro pertença ao interior do agrupamento.)

A lista de agrupamentos satisfaz uma propriedade fundamental: cada objecto x da base de dados está guardado no primeiro agrupamento que o pode conter, ou seja, se a lista for iterada, x pertence ao primeiro agrupamento (c, r, I) que verificar $d(x, c) \leq r$. Para completar a definição de RLC [7], é necessário especificar dois parâmetros: um real positivo ρ e um inteiro positivo α. O primeiro determina o raio de todos os agrupamentos da lista e o segundo permite definir a implementação dos interiores. Sempre que o número de objectos no interior de um agrupamento não exceder α, o interior é designado por *folha* e está implementado num vector. Nos restantes casos, o interior é uma RLC (com raio ρ e folhas com capacidade α).

Repare que, como o interior de um agrupamento pode ser uma lista de agrupamentos, um objecto pode pertencer a vários agrupamentos, organizados hierarquicamente, aos quais se atribui uma *profundidade*. Considera-se que os agrupamentos da lista principal têm profundidade zero.

Para minimizar o número de distâncias calculadas nas pesquisas (como veremos a seguir), guarda-se, para cada objecto, uma sequência com as distâncias do objecto aos centros dos agrupamentos a que ele pertence, chamada a *sequência de distâncias* do objecto. Portanto, o centro de um agrupamento de profundidade p está associado a p distâncias e cada objecto de uma folha que é o interior de um agrupamento de profundidade p tem uma sequência com $p+1$ distâncias.

A Fig. 1 esquematiza uma RLC com raio 3 e folhas com capacidade 5. Os símbolos c, r, p, $|I|$ e I denotam, respectivamente, o centro, o raio, a profundidade, o número de objectos no interior e o interior do agrupamento. Por exemplo, um objecto w no interior do agrupamento de centro z_1 pertence a três agrupamentos, cujos centros são x_1, y_2 e z_1. A sequência de distâncias de w é constituída por $d(w, x_1)$, $d(w, y_2)$ e $d(w, z_1)$.

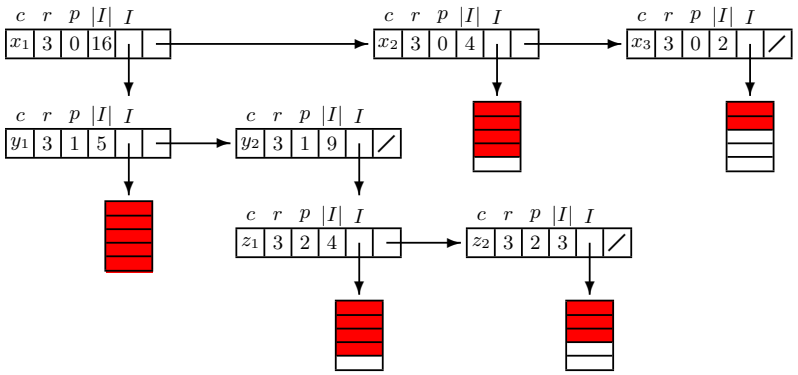


Figura 1. Exemplo de RLC com raio 3 e folhas com capacidade 5. Indica-se o raio e a profundidade de cada agrupamento para clarificar estas noções. Mas, para simplificar a figura, omitem-se todas as sequências de distâncias.

3.2 Descrição Sucinta dos Algoritmos

O próximo objectivo é descrever os grandes passos dos algoritmos sobre a RLC, estando os detalhes especificados em [7].

O carregamento inicial da RLC é feito inserindo sucessivamente cada um dos objectos. Para inserir um objecto x (que, para simplificar, se assume não se encontra na RLC), a lista é iterada até se encontrar um agrupamento (c, ρ, I) que possa conter x (i.e., tal que $d(x, c) \leq \rho$).

- Se nenhum agrupamento for encontrado, cria-se um novo agrupamento (com centro x e interior vazio), que é adicionado à cauda da lista.
- No caso contrário, x é inserido em I.
  - Quando I é uma folha, se há espaço no vector para mais um elemento, x é aí inserido. Se o vector está cheio, transforma-se I numa RLC, criando uma lista de agrupamentos vazia, onde se insere x e cada um dos elementos presentes na folha.
  - Se I é uma lista de agrupamentos, insere-se (recursivamente) x em I.

A remoção é semelhante. Mas não será explicada, por falta de espaço, uma vez que os testes experimentais aqui descritos não envolvem remoções.

Na pesquisas por proximidade, itera-se a lista, identificando-se a relação entre cada agrupamento (c, ρ, I) e a interrogação (q, r), com base na distância entre c e q e no valor dos raios ρ e r. Escusado será dizer que c pertence ao conjunto resposta se $d(c, q) \leq r$. Existem basicamente quatro situações distintas.

- Se a região da interrogação está contida na região do agrupamento, a pesquisa prossegue pelo interior do agrupamento e a iteração pára.
- Se a região da interrogação contém a região do agrupamento, todos os objectos do interior do agrupamento são imediatamente adicionados ao conjunto resposta e a iteração continua.

– Se as regiões da interrogação e do agrupamento se intersectam (mas nenhuma contém a outra), a pesquisa prossegue pelo interior do agrupamento e a iteração continua.
– Se as regiões da interrogação e do agrupamento são disjuntas, o interior do agrupamento é ignorado e a iteração continua.

Repare que se podem incluir ou excluir do conjunto resposta todos os objectos do interior de um agrupamento, sem se ter calculado qualquer distância entre eles e o ponto da interrogação. Quando a pesquisa chega a uma folha, também é possível concluir que um objecto x deve, ou não deve, ser colocado no conjunto resposta, sem se calcular a sua distância a q. Para isso, usam-se as distâncias $d(c, q)$ e $d(x, c)$, onde c é o centro de um agrupamento ao qual x pertence. Fazendo $m = d(c, q) - r$, prova-se que:

– se $d(x, c) < m$, então $d(x, q) > r$ (e x não pertence à resposta);
– se $d(x, c) \leq -m$, então $d(x, q) \leq r$ (e x pertence à resposta).

Note que $d(c, q)$ teve de ser calculado para se entrar no interior do agrupamento e que $d(x, c)$ é um dos elementos da sequência de distâncias de x. A distância entre x e q só é calculada quando, para todo o agrupamento a que x pertence, nenhuma das duas regras pode ser aplicada.

3.3 Implementação em Memória Secundária

A RLC foi implementada em C++.[1] Para reduzir o número de acessos a ficheiro, a informação está guardada em páginas de dimensão fixa (por exemplo, com 4 096 ou 8 192 bytes). Há, basicamente, três tipos de páginas.

– O *cabeçalho* tem informação global da estrutura e do tipo de objectos.
– As *páginas de agrupamentos* permitem implementar as listas de agrupamentos através de listas ligadas de vectores de agrupamentos. Por cada agrupamento, guarda-se o centro, a sequência das distâncias do centro, o número de pontos no interior do agrupamento e o apontador para a (primeira) página onde se encontra o interior do agrupamento.
– De modo semelhante, as *páginas de folhas* permitem implementar cada folha através de uma lista ligada de vectores de pares, cuja primeira coordenada é um objecto e a segunda é a respectiva sequência de distâncias.

Existem três restrições adicionais: não há vectores (de agrupamentos ou de pares) vazios; não existem posições vazias entre os elementos presentes num vector; e nenhum elemento de um vector pode ocupar mais do que uma página.

Esta última restrição impediu que se usassem páginas de dimensão razoável com algumas bases de dados, porque a RLC atingia profundidades tão elevadas que os pares das folhas mais profundas ocupavam demasiada memória. A grande profundidade da RLC deve-se aos raios dos agrupamentos serem todos iguais, como se ilustra na Fig. 2, com pontos no plano e a distância euclidiana.

[1] A primeira implementação da RLC em memória secundária, que só aceitava pontos em $\mathbb{R}^n$ com a distância euclidiana, foi realizada por Carlos Rodrigues [9].

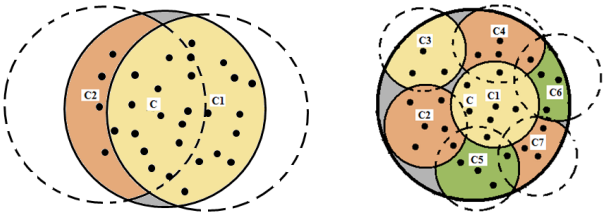


Figura 2. Interior de um agrupamento com centro c. *(Esquerda)* O raio dos agrupamentos é fixo. *(Direita)* O raio dos agrupamentos diminui com a profundidade.

Quando um agrupamento tem muitos pontos, o seu interior I é uma lista de agrupamentos. Ora, quaisquer que sejam os centros dos primeiros agrupamentos dessa lista (c_1 e c_2, do lado esquerdo da Fig. 2), as regiões que eles definem são tão grandes que contêm quase todos os objectos de I. De facto, quando o raio dos agrupamentos é fixo, as listas de agrupamentos (de profundidade positiva) têm um comprimento muito reduzido e a estrutura tem uma profundidade elevada. A ideia da segunda implementação da RLC em memória secundária [11] consiste na redução progressiva do raio dos agrupamentos, à medida que a profundidade aumenta (como exemplificado do lado direito da Fig. 2). Mais precisamente, a RLC continua a ter apenas dois parâmetros, o raio ρ e a capacidade α das folhas, mas o raio de cada agrupamento depende da profundidade p do agrupamento, sendo dado por $\frac{\rho}{p+1}$. Nesta variante, a distribuição dos objectos de um interior pelos agrupamentos do nível seguinte, não só envolve mais agrupamentos, como também é bastante mais equilibrada. Consequentemente, a RLC cresce significativamente em largura, mantendo profundidades muito baixas.

4 Espaços Métricos

O maior desafio das estruturas de dados métricas é terem bons desempenhos com qualquer espaço métrico. O problema é que as distâncias entre os objectos de um universo podem ter distribuições muito diferentes e essa distribuição tem impacto na forma das estruturas de dados e na eficiência dos algoritmos.

Foram seleccionados quatro espaços métricos com dados reais. Dois universos são dicionários[2], tendo sido usada a distância de Levenshtein (3). O dicionário de alemão tem 74 916 palavras, cujos comprimentos variam entre um e trinta e três, e o dicionário de inglês tem 69 069 palavras com comprimentos entre um e vinte e um. O terceiro espaço métrico é composto por histogramas de imagens[3] e pela distância euclidiana (2). Cada um dos 112 543 histogramas é uma sequência de cento e doze números reais. O último universo (cedido por Pedro Chambel [1]) é constituído por 3 040 vectores, cada um com vinte e quatro valores reais que sintetizam características extraídas de imagens de faces humanas. A métrica para as imagens de rostos é a de Manhattan (1).

[2] Os ficheiros com as palavras foram obtidos em `http://www.sisap.org/` .
[3] O ficheiro foi retirado de `http://www.dbs.informatik.uni-muenchen.de/` .

Para conhecer algumas propriedades dos espaços métricos, calcularam-se, para cada universo, todas as distâncias entre dois objectos distintos. A Fig. 3 apresenta os histogramas com as distribuições dessas distâncias e mais algumas estatísticas. No caso dos dois espaços métricos de imagens, como as distâncias são reais, foram agrupadas em vinte e cinco intervalos de igual dimensão.

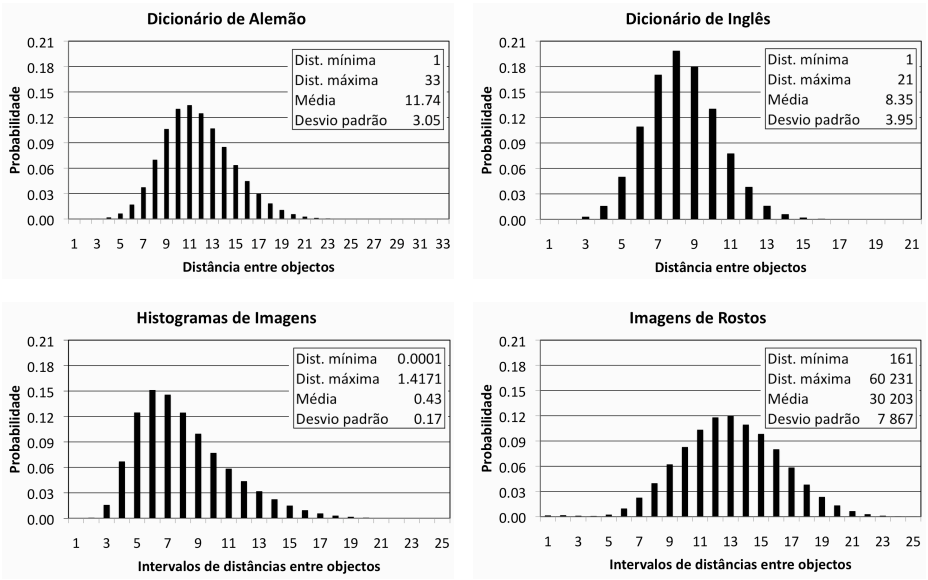


Figura 3. Histogramas das distâncias para os quatro espaços métricos.

5 Resultados Experimentais

A RLC foi comparada com três estruturas de dados métricas genéricas, implementadas em memória secundária: M-tree [2], Slim-tree [15] e DF-tree [16]. Usaram-se as implementações da biblioteca GBDI Arboretum [5], escrita em C++. Não se testaram mais estruturas de dados devido à inexistência de implementações de domínio público.

Todas as estruturas de dados são parametrizadas. Para o parâmetro comum, que é a dimensão das páginas do ficheiro em bytes, foi escolhido o valor 4 096. Em relação à M-tree, usaram-se os métodos mais eficazes: o do hiperplano generalizado para particionar os nós e o que minimiza a soma dos raios para efectuar as promoções. A implementação da Slim-tree permite definir a sub-árvore onde se insere um objecto, quando mais do que uma o pode conter, e o método de particionamento dos nós. As escolhas, aconselhadas em [15], recaíram, respectivamente, sobre o método que selecciona a sub-árvore cuja raiz tem menor ocupação e sobre o particionamento induzido pela árvore mínima de cobertura. A DF-tree

tem quatro parâmetros, dos quais dois são os da Slim-tree, que seleccionaram os mesmos algoritmos. Para o número de representantes globais, recorreu-se à dimensão máxima dos pontos do universo: 33 para o dicionário de alemão, 21 para o de inglês, 112 para os histogramas de imagens e 24 para as imagens de rostos. Por último, o valor a partir do qual o conjunto de representantes globais é actualizado foi o utilizado pelos autores nos seus testes experimentais [16]: 2 000. Os parâmetros da RLC, determinados por observação de resultados experimentais [11], variam com o espaço métrico. O raio usado foi 10, 6, 0.71 e 23 160, e a capacidade das folhas foi 100, 100, 50 e 25, respectivamente, para os dicionários de alemão e de inglês, os histogramas de imagens e as imagens de rostos.

Das quatro estruturas de dados, a RLC é a única que suporta a operação de remoção. Por esse motivo, os testes experimentais relatados neste artigo só avaliam inserções e pesquisas (por proximidade).

Foram gerados quatro ficheiros por cada espaço métrico. Três são permutações aleatórias do universo e foram usados para carregar a base de dados por ordens diferentes. O outro, com cerca de 10% dos objectos do domínio (também seleccionados aleatoriamente), contém as interrogações. Os raios das interrogações foram gerados uniformemente dentro do intervalo $[1, 3]$ para os dicionários, $[0.00001, 0.1]$ para os histogramas de imagens e $[5\,000, 15\,000]$ para as imagens de rostos. Para cada espaço métrico, cada teste consistiu exactamente nas mesmas inserções e nas mesmas pesquisas, variando apenas a ordem pela qual os objectos foram inseridos, que tem influência na forma final das estruturas de dados. Todos os resultados apresentados são a média dos valores obtidos nos três testes do mesmo universo.

A Fig. 4 apresenta o número médio de distâncias calculadas por operação de inserção e de pesquisa, o número médio de leituras de ficheiro efectuadas por operação de inserção e de pesquisa, e o número médio de operações de escrita em ficheiro por operação de inserção. Para os valores das distâncias, dividiram-se os respectivos números médios pelo número de objectos do universo, para se compreender melhor o desempenho das estruturas de dados. Repare que, se essa percentagem for 70% (que é o valor para a pesquisa na Slim-tree com o dicionário de inglês), o ganho face ao algoritmo que percorre um vector com os objectos da base de dados e calcula as distâncias entre todos os elementos do vector e o ponto da interrogação é de apenas 30%.

Numa primeira análise, pode-se concluir que os valores mínimos estão sempre associados à M-tree, à Slim-tree ou à RLC. De facto, a RLC não tem o melhor desempenho em apenas cinco dos vinte casos: no número de distâncias por inserção com os dicionários de alemão (0.12% na Slim-tree e 0.50% na RLC) e de inglês (0.14% na Slim-tree e 0.36% na RLC) e no número de leituras por inserção com o dicionário de alemão (3 na M-tree e na Slim-tree; 6 na RLC) com o dicionário de inglês (2.9 na M-tree e na Slim-tree; 3.4 na RLC) e com os histogramas de imagens (11 na Slim-tree e 15 na RLC). No entanto, sempre que os valores da RLC não são os mais baixos, as diferenças são pouco significativas. Em todos os outros casos, e, em particular, nos dois parâmetros da avaliação das pesquisas e no número de escritas em ficheiro, a RLC é imbatível.

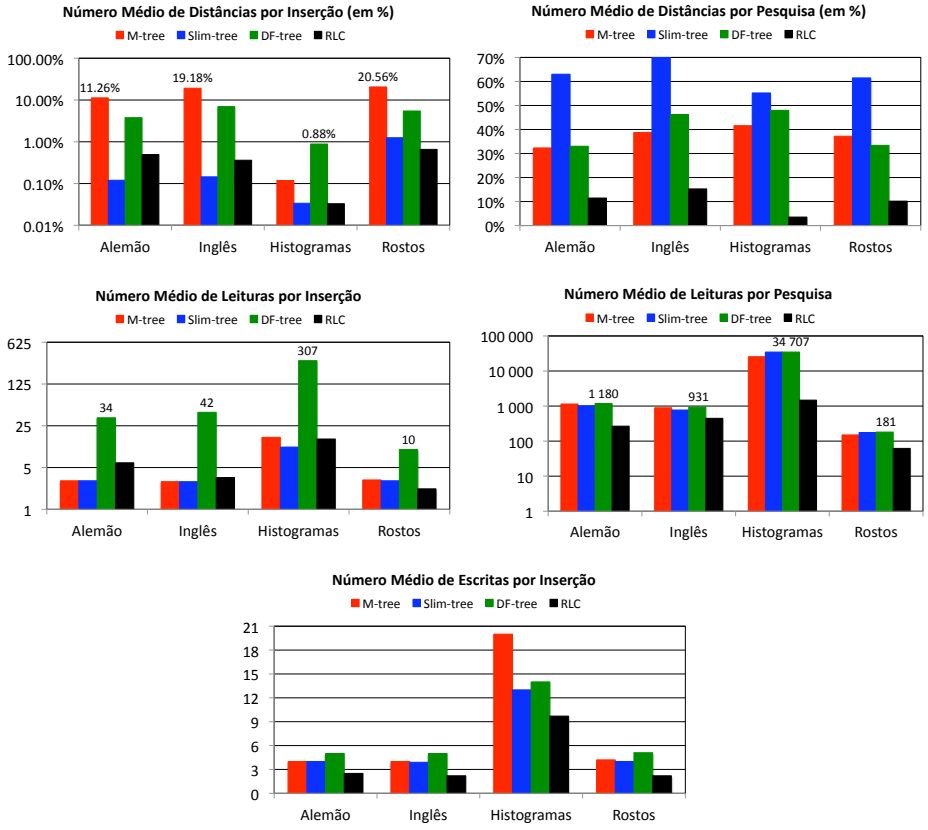


Figura 4. Números médios de distâncias calculadas entre objectos, de leituras e de escritas em ficheiro, por inserção e por pesquisa. Nos dois gráficos de cima, os valores correspondem à percentagem do número médio de distâncias calculadas em relação ao número de objectos da base de dados. Note que a escala de três gráficos é exponencial, para que todas as barras sejam visíveis. Nesses casos, apresenta-se o valor da barra mais alta para cada espaço métrico.

Estes resultados ainda são mais favoráveis para a RLC do que aqueles que se obtiveram ao comparar estruturas de dados métricas em memória central [6,7,8,1,3]. Note que as diferenças nos números de distâncias calculadas nas pesquisas são impressionantes. Acresce que a RLC é verdadeiramente dinâmica. Portanto, ao contrário do que é usual quando se comparam estruturas de dados, neste caso não é necessário optar entre um bom desempenho nas pesquisas e nas inserções e a possibilidade de se efectuar uma operação tão importante como a remoção.

Na Tabela 1, apresentam-se alguns dados sobre a forma da RLC, obtidos com um dos ficheiros de teste. Verifica-se que há interiores implementados em listas com muitos agrupamentos e que a profundidade da estrutura é pequena.

Tabela 1. Dados sobre a forma da RLC com todos os objectos do espaço métrico.

Espaço métrico	Número de agrupamentos	Comprimento da lista principal	Comprimento máximo de outra lista	Profundidade máxima
Alemão	22 976	2 360	2 494	4
Inglês	18 559	2 281	1 033	4
Histogramas	25 747	91	422	9
Rostos	326	42	67	2

6 Conclusões e Trabalho Futuro

Descreveu-se e avaliou-se uma adaptação da estrutura de dados RLC a memória secundária. A RLC revelou-se equivalente à Slim-tree nas inserções, com a qual partilhou os melhores desempenhos, e consideravelmente mais eficiente que todas as outras estruturas de dados analisadas nas pesquisas por proximidade.

Em relação ao trabalho futuro, pretende-se continuar a avaliar a RLC, estendendo os testes experimentais a dados de outras áreas (como biologia computacional ou sistemas de informação geográfica) e a mais implementações de estruturas de dados métricas. É importante obter versões da Slim-tree e da DF-tree que incorporem o algoritmo *slim-down* [15] (que não está implementado na biblioteca Arboretum) e implementações de estruturas de dados que suportem a operação de remoção.

Para minorar o problema da parametrização da RLC, pretende-se descobrir fórmulas para o raio e para a capacidade das folhas que dependam de características do espaço métrico. Como o impacto do raio no desempenho da estrutura é enorme, e o impacto da capacidade das folhas é bastante reduzido, começou-se pelo primeiro, tendo-se chegado à fórmula $\rho = \mu - \frac{\sigma}{2}$, onde μ é a média e σ é o desvio padrão da distribuição das distâncias. Parece que os valores obtidos (c.f. Tabela 2) são uma boa aproximação para distribuições normais, não se aplicando à distribuição dos histogramas de imagens.

Tabela 2. Valores da fórmula para o raio, para distribuições normais.

	Alemão	Inglês	Histogramas	Rostos
Média (μ)	11.74	8.35	0.43	30 203
Desvio padrão (σ)	3.05	3.95	0.17	7 867
Raio usado	10	6	0.71	23 160
Valor de $\mu - \frac{\sigma}{2}$	10.22	6.38	0.35	26 270

Agradecimentos Ângelo Sarmento teve uma bolsa semestral concedida pelo Centro de Informática e Tecnologias da Informação (CITI), hospedado no Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. Os autores agradecem a Luís Caires, por ter disponibilizado

uma máquina eficiente para a realização dos testes experimentais, a Carlos Rodrigues, pela ajuda na utilização da biblioteca Arboretum, e a Pedro Chambel, pela cedência dos dados das imagens de rostos.

Referências

1. Chambel, P.: Pesquisa de Imagens de Rosto. Master's thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (2009), http://citi.di.fct.unl.pt/
2. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: 23rd Int. Conf. on Very Large Data Bases (VLDB'97). pp. 426–435. Morgan Kaufmann, San Francisco (1997)
3. Costa, F.: Geração automática de playlists de músicas semelhantes. Master's thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (2009), http://citi.di.fct.unl.pt/
4. Dohnal, V., Gennaro, C., Savino, P., Zezula, P.: D-Index: Distance searching index for metric data sets. Multimedia Tools and Applications 21(1), 9–33 (2003)
5. GBDI Arboretum, http://www.gbdi.icmc.usp.br/arboretum/
6. Mamede, M.: Recursive lists of clusters: A dynamic data structure for range queries in metric spaces. In: 20th Int. Symp. on Computer and Information Sciences (ISCIS 2005). LNCS, vol. 3733, pp. 843–853. Springer, Heidelberg (2005)
7. Mamede, M.: A dynamic data structure for range queries in high dimensional metric spaces. Tech. rep., Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (2007), http://ctp.di.fct.unl.pt/~mm/dynamic-07.pdf
8. Mamede, M., Barbosa, F.: Range queries in natural language dictionaries with recursive lists of clusters. In: 22nd Int. Symp. on Computer and Information Sciences (ISCIS 2007). pp. 1–6. IEEE CS Press, Los Alamitos, CA (2007)
9. Rodrigues, C.: Implementação de sistemas de indexação para espaços métricos. Relatório do projecto final de curso, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (2006)
10. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Computer Graphics and Geometric Modeling, Morgan Kaufmann, San Francisco (2006)
11. Sarmento, A.: Estruturas de Dados Métricas Genéricas em Memória Secundária. Master's thesis, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (2010), http://citi.di.fct.unl.pt/
12. Sexton, A.P., Swinbank, R.: Symmetric M-tree. Tech. Rep. CSR-04-02, School of Computer Science, University of Birmingham (2004), http://www.cs.bham.ac.uk/~rjs/research/
13. Thomasian, A., Zhang, L.: Persistent semi-dynamic ordered partition index. The Computer Journal 49(6), 670–684 (2006)
14. Thomasian, A., Zhang, L.: The stepwise dimensionality increasing (SDI) index for high-dimensional data. The Computer Journal 49(5), 609–618 (2006)
15. Traina, Jr., C., Traina, A., Faloutsos, C., Seeger, B.: Fast indexing and visualization of metric data sets using Slim-trees. IEEE Transactions on Knowledge and Data Engineering 14(2), 244–260 (2002)
16. Traina, Jr., C., Traina, A., Filho, R.S., Faloutsos, C.: How to improve the pruning ability of dynamic metric access methods. In: 11th Int. Conf. on Information and Knowledge Management (CIKM'02). pp. 219–226. ACM Press, New York (2002)
17. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Advances in Database Systems, Springer (2006)

LiveWeb – Core Language for Web Applications

Miguel Domingues and João Costa Seco

CITI - Departamento de Informática FCT/UNL, Lisboa, Portugal
miguel.domingues@fct.unl.pt, joao.seco@di.fct.unl.pt

Abstract. We present a typed core language for web applications that integrates interface definition, business logic and database manipulation. By expressing the interactions between different application layers in the same programming language, we gain the benefits of strongly typed languages without loosing the programming flexibility of interpreted languages which is frequently an argument in favor of unsafe programming languages. We describe a prototype of a programming environment and runtime support system for our language that allows a very dynamic style of web application development.

Keywords: Web applications, programming languages, type systems

1 Introduction

The main-stream of web application development is usually based on a three-layer architecture that divides applications into client interface, business logic and database layers. In practice, applications are developed in heterogeneous programming language environments, and in particular, the application logic is specified using general purpose programming languages to define computations and specialized query languages to access the information stored in databases. The two language paradigms have several mismatches [5,6] making the integration between layers one of the most important aspects of web application development. Typically, layers interact through dialects and programming conventions, and communication code is not subject to effective mechanical verification and is highly error prone. Writing SQL queries as strings is a simple and fast way to implement database applications but doesn't allow for any kind of static checks. Object-Relational Mapping approaches provide a safer solution to this problem but are in many cases considered too heavy [11].

Furthermore, web application development is very high demanding for rapid construction and constant change, which gave rise to a series of flexible languages that trade the benefits of statically strongly typed programming languages for the advantages of dynamically typed interpreted languages (e.g., PHP, ASP, Ruby). Some development frameworks targeting web applications (e.g., Ruby On Rails, CakePHP) provide scaffolding features to increase developers' productivity, others provide extensions to general purpose languages and include typing for database operations [3,7], a third category of frameworks choose to use domain specific languages to provide program safety by construction [1,4,7,14]. We

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 91–94

argue that the latter approach, to integrate query support in the language and hence enable static verification between layers, has clear advantages and is more challenging from a programming language perspective.

Although the rising of the level of abstraction allows for checking the basic safety of programs and elimination of many programming errors, our language aims at potentiating the verification of other more sophisticated properties. In particular, we refer to properties related to data security and access control [2,10,12] and related to the coordination of several interacting parts in distributed systems [13]. In order to allow future experiments on these theoretical studies on type systems we introduce a typed core language for web applications that integrates the typing of interface definition, business logic and database manipulation. A more complete description of the language is available in [8]. Our approach compares to Links [7] and Ur/Web [4] that also define strongly typed languages but we take a more limited approach of starting from first principles with primitive operations, types and a clear separation between interface and program, and thus allowing for formal studies to be easily applied here.

We also describe an implementation of an interpreter and a highly flexible programming environment for our language, designed to provide a dynamic programming style where the developers act directly over the actual running code without loosing the global integrity checks of interpreted languages.

2 Core Language for Web Applications

Our core language has three main programming elements: entities, screens and actions. Entities are containers of structured persistent data implemented in database tables. Operations over entities mimic a subset of the standard database query language (SQL). Screens are abstractions over a user interface definition language whose values are web pages. Screens may be parameterized and some of the user interface expressions may contain general purpose expressions to be executed back at the server. Actions are abstractions over general purpose expressions comprising operations over entities, screens and other values.

We now illustrate the syntax of the language by means of the code fragment in Fig. 1 implementing a phone number directory. We define an entity called Person containing phone numbers and names by enumerating its attributes and corresponding types. The interface of the application is defined in a screen called directory, built by iterating the results of a **from** expression (written in a syntax similar to LINQ [3]) that fetches all values stored in entity Person. The language fragment used to define web pages is inspired in the nested structure of web page blocks, it contains an **iterator** expression that maps query results in web page blocks, and also contains input elements (**textfield**) and actuator elements (**button**). Input elements declare local variable names that can be used in expressions that pass the control flow from the browser back to the web server. The **button** element in screen directory calls action addPerson using as arguments the values given by the user in the text fields, which are available through to the local names name and phone. Action addPerson adds a new row to entity

```
def entity Person { id:Id, name:String, phone:String }
def screen directory {
  iterator (row in (from (p in Person) select p)) {
    label "Name: " + row.name; br;
    label "Phone: " + row.phone; br; br
  };
  label "Name"; textfield name; br;
  label "Phone"; textfield phone; br;
  button "Add" to addPerson(name,phone)
}
def action addPerson(nm:String, ph:String):Block {
  insert { name = nm, phone = ph } in Person;
  directory()
}
```

Fig. 1: LiveWeb Example

Person with the given values for name and phone. After the insertion of a new row, screen directory is rendered in the browser. The type system of the language ensures that there are no runtime errors due to ill-formed queries, with missing entity names or ill-typed arguments in where clauses, it ensures that screens are rendered properly, e.g. with no missing information from entity attributes, and that all actions used in screens exist and expect matching parameter types, etc.

3 Runtime Support System

We implement our language in a runtime support system that combines a web server with a language interpreter and a database for application data. The system also provides a wiki style development environment based on a persistent and versioned code base. Source code is stored, versioned, and organized in a database instead of being scattered in files. This allows for the type safe dynamic reconfiguration of the system, since we maintain as active the code that was last verified as well typed. The UI fragment evaluates to regular HTML code with JavaScript for name binding and activating continuation code in the server.

Our runtime support system provides two functioning modes, one for executing the application and another for editing and checking the source code. The first mode of interaction, the *execution mode*, allows for actions and screens to be called by standard URL conventions using the name of element (action or screen) and by indicating the corresponding arguments by means of literals. More complex browser interactions can be achieved by standard techniques but are out of the scope of this work. Query expressions are evaluated to regular SQL expressions and executed in the database.

The *development mode* is also available through the browser (usually through an "edit" button or link). It lets the user access and change all available elements of an application. The runtime support system stores screen, action, and entity definitions as separate pieces of code, and establishes a notion of published version of an application built from the latest verified copies. When the structure

of an entity is modified, the database model of application data must also be modified to match the new entity definition. For the sake of simplicity, entity data is transformed in the more direct way in order to keep applications working.

4 Final Remarks

On the one hand, this work aims at developing a simple and small language that could be easily extended and allow the formal study of type related properties like data security and access control. On the other hand, it aims at providing an implementation of a runtime system for the language that works like a workbench for those extensions. Security related property checking techniques based on refinement types [9] are presented in [2] and a prototype is already available from the authors' web page. There are many technological issues that can be improved to make the language and runtime system more usable (e.g. nested queries, lazy query evaluation, asynchronous page updates, etc.) and robust (keeping versions of data of deleted columns, etc.). However, the main goal is to provide a framework to future experiments on type systems for web applications.

Acknowledgments. This work is partially supported by the Certified Interfaces project NGN44-CMUPortugal. We thank to Luís Caires, António Melo and Lúcio Ferrão for the discussions at OutSystems that motivated this work.

References

1. OutSystems (Jan 2010), `http://www.outsystems.com/`
2. Caires, L., Perez, J.A., Seco, J.C., Vieira, H.T.: Refinement Types for Database Access Control. Tech. rep., UNL-DI-3-2010, Dep. Informática, FCT/UNL (2010)
3. Calvert, C., Kulkarni, D.: Essential LINQ. Addison-Wesley Professional (2009)
4. Chlipala, A.: Ur: Statically-Typed Metaprogramming with Type-Level Record Computation. PLDI 2010, SIGPLAN Notices 45(6), 122–133 (2010)
5. Cook, W.R., Ibrahim, A.H.: Integrating Programming Languages and Databases: What's the Problem? In: ODBMS.ORG, Expert Article (2005)
6. Cooper, E.: The Script-Writer's Dream: How to Write Great SQL in Your Own Language, and Be Sure It Will Succeed. DBPL 2009, LNCS 5708 (2009)
7. Cooper, E., Lindley, S., Wadler, P., Yallop, J.: Links: Web programming without tiers. FMCO 2006, LNCS 4709, 266–296 (2006)
8. Domingues, M., Seco, J.C.: Definition of a Core Language for Web Applications (LiveWeb). Tech. rep., UNL-DI-4-2010, Dep. Informática, FCT/UNL (2010)
9. Freeman, T., Pfenning, F.: Refinement Types for ML. PLDI 1991, SIGPLAN Notices 26(6) (1991)
10. Pires, M., Caires, L.: A type system for access control views in object-oriented languages. In: ARSPA-WITS 2010. LNCS (2010)
11. Spiewak, D., Zhao, T.: ScalaQL: Language-Integrated Database Queries for Scala. SLE 2009, LNCS 5969, 154–163 (2010)
12. Toninho, B., Caires, L.: A spatial-epistemic logic and tool for reasoning about security protocols. Tech. rep., Dep. de Informática, FCT/UNL (2009)
13. Vieira, H.T., Caires, L., Seco, J.C.: The conversation calculus: A model of service oriented computation. ESOP 2008, LNCS 4960 (2008)
14. Visser, E.: WebDSL: A case study in domain-specific language engineering. GTTSE II, LNCS 5235 (2008)

Replicated Software Components for Improved Performance*

Paulo Mariano, João Soares and Nuno Preguiça

CITI / Dep. de Informática - Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa,
Quinta da Torre, 2829 -516 Caparica, Portugal

Abstract. In recent years, CPU evolution has shifted from the continuous increase of speed to an increase in the number of processing cores. In this paper, we propose to take advantage of the new multicore systems, by diverse replication of software components. This approach, complementary to other directions that are being tackled, attempts to obtain a better performance for a *macro-component* consisting of several diverse implementations of the same specification, by returning, for each operation, the result from the replica that is faster for that operation. We present an early design of a system that implements this approach.

Keywords: replication, parallel programming, multicore systems

1 Introduction

Until recently, CPUs evolution was a mix of improved functionality and a steadily increase of clock speed. However, this path of evolution have reached its end, with an increasing difficulty on further increasing clock speed [4]. Currently, hardware manufacturers are deploying CPUs with an increasing number of processing cores. With multicore CPUs, programs must include multiple concurrent threads of activity to take benefit from the multiple cores available.

Previous experience in the field of concurrent/parallel programming shows that creating such applications is a highly demanding task even for experienced programmers. This problem has led to an intense research activity for finding good abstractions for expressing parallel computations and to build suitable runtime support [2,5,3] that simplifies this task in multicore environments.

In this paper, we propose a complementary approach that can be used by both applications that include multiple threads and by applications that include a single thread. The main insight for our approach is that applications almost always resort on a set of components with standard interfaces - e.g. data structures, algorithms, etc. For these components, several implementations are available, which have different performance for different inputs or for different

* This work was partially supported by FCT/MCTES, project #PTDC/EIA/74325/2006 and #PTDC/EIA-EIA/108963/2008 and CITI.

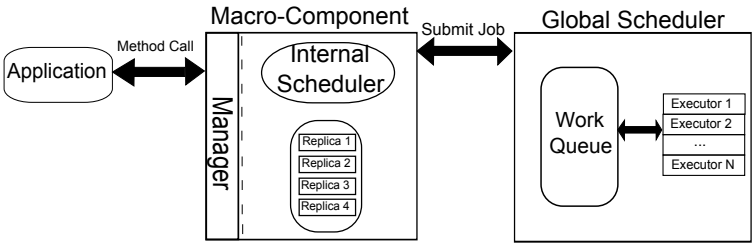


Fig. 1. Macro-component system model.

operations. Thus, we propose to locally replicate these components using different underlying implementations, building a *macro-component*.

This basic macro-component concept can be used for two different purposes. Reliability, by masking software bugs, similar to n-version programming [1] but on a smaller scale. Or improved performance, by returning the result obtained by the fastest replica. In this case, an operation would be executed in the underlying replicas concurrently and the first result returned by these would be the result returned by the macro-component.

Although the idea seems simple, making it work in practice has a number of technical challenges that must be addressed, including performance issues, coordination among macro-components in an application, etc. In this paper we present our initial design for building and supporting *macro-components*.

The remaining of this paper is organized as follows: Section 2 describes our initial design and Section 3 concludes the paper with some final remarks.

2 Design

In this section we present the initial design for a macro-component runtime system. To simplify this initial design a number of assumptions are made about the macro-component underlying micro-components (or replicas). First, it is assumed that the replicas do not fail arbitrarily. Second, they must be self contained. Implementations are not allowed to interact with their exterior through any means other than operation results. Third, operations must be deterministic.

In figure 1, we present the macro-component runtime architecture. The architecture can be divided in two main components: the global scheduler and the individual macro-components. The macro-components are composed of a manager, which provides the interface of the implemented specification to the exterior, a set of underlying replicas and an internal scheduler. It is not necessary to have any knowledge on the implementation details of the underlying replicas as long as they all implement the same specification. The global scheduler keeps track of the several execution jobs in a work queue and schedules them for execution by a pool of *executor* threads in a producer-consumer scheme. More details on the purpose and functioning of both the internal schedulers and the global scheduler can be found in section 2.1.

When the application calls an operation on the macro-component, the manager forwards it to the internal scheduler. The internal scheduler creates one or more *jobs* for the operation which are then submitted to the global scheduler. If the operation can be executed asynchronously (i.e., it returns no result and can never fail), the application thread can immediately continue execution, otherwise it must block until a result is available. Operations submitted to the global scheduler are kept in a work queue until they are handed for execution to one of the executor threads.

2.1 Scheduling

The scheduling of operations in a macro-component environment takes two forms in the proposed model, *internal* scheduling and *global* scheduling. Internal scheduling is done internally in the macro-component and consists on the creation of *jobs* to be submitted to the global scheduler. These jobs consist on the execution of an operation on a given replica. As such, the internal scheduler is responsible to decide which replicas will run an operation. Global scheduling, on the other hand, handles the choice of which *job* to execute at any given moment.

Internal Scheduling As replica states must be maintained coherent, operations which update this state must be executed everywhere. However, for read operations this is not required. In our prototype we have implemented the following three internal scheduling strategies based on this property.

Read-all Reads are executed in all replicas and the result returned will be that of the replica that finishes processing first. If the macro-component experiences a light load, this strategy ensures the best performance for all operations. However, stress tests reveal a problem for this strategy as replicas can be held back by unnecessary slow operations instead of useful work.

Read-one With read-one, the macro-component directs a read operation to a single replica. This replica is, hopefully, the one with the best performance for this operation type. The issue with this strategy is how can we predict which replica best fits an operation.

Read-multiple A compromise between read-all and read-one modes. An operation is assigned to a *subset* of the replicas.

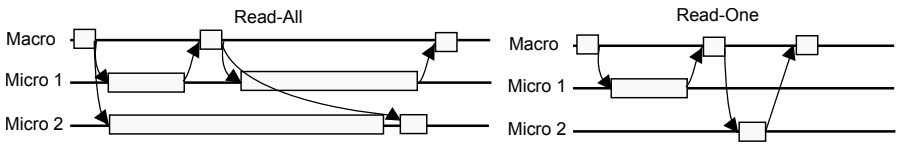


Fig. 2. Read-all vs read-one example timeline.

Global Scheduling This scheduler works as a distributor of jobs for the executors. Various scheduling strategies can be implemented, but some concerns must be addressed in order to maintain proper functionality. First, the relative order of update operations must be preserved. This means that update operations must be executed in the same order on all replicas. If these operations are executed out of order, macro-component replicas' internal state may diverge, leading to incorrect or unexpected results. Also, while read operations can be reordered they must be executed in a state that reflect all previous updates.

For simple independent macro-components, these ordering rules can be enforced for each macro-component independently, as there is no relationship between the replicas. In more complex cases, this independence may not hold due to relationships between macro-components. For example, in a macro-component based JDBC driver, the Connection macro-component cannot be allowed to commit before all previous operations on its Statements are finished.

3 Final Remarks

In this paper, we introduce the concept of macro-component, a software component that combines several different implementations of a given component specification. We propose the use of macro-components as a mechanism to improve the performance of applications in multicore systems and present an initial design of a runtime system to support this concept.

The initial evaluation with an early prototype of our system shows that the runtime system imposes non-negligible overhead. However, even with a non-optimized implementation it was possible to obtain a better overall performance for a *macro-component* that implements an in-memory SQL database.

References

1. A. Avizienis. The n-version approach to fault-tolerant software. *IEEE Trans. Softw. Eng.*, 11(12):1491–1501, 1985.
2. J. Larus and C. Kozyrakis. Transactional memory. *Commun. ACM*, 51(7):80–88, 2008.
3. E. B. Nightingale, D. Peek, P. M. Chen, and J. Flinn. Parallelizing security checks on commodity hardware. In S. J. Eggers and J. R. Larus, editors, *ASPLOS*, pages 308–318. ACM, 2008.
4. K. Olukotun and L. Hammond. The future of microprocessors. *Queue*, 3(7):26–29, 2005.
5. C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *HPCA '07: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society.

Compiladores e Linguagens de Programação

A Static Approach for Detecting Concurrency Anomalies in Transactional Memory

Bruno Teixeira, João Lourenço, and Diogo Sousa*

CITI — Departamento de Informática,
Universidade Nova de Lisboa, Portugal
bct18897@fct.unl.pt Joao.Lourenco@di.fct.unl.pt
dm.sousa@fct.unl.pt

Abstract. Programs containing concurrency anomalies will most probably exhibit harmful erroneous and unpredictable behaviors. To ensure program correctness, the sources of those anomalies must be located and corrected. Concurrency anomalies in Transactional Memory (TM) programs should also be diagnosed and fixed. In this paper we propose a framework to deal with two different categories of concurrency anomalies in TM. First, we will address *low-level TM anomalies*, also called dataraces, which arise from executing programs in weak isolation. Secondly, we will address *high-level TM anomalies*, also called high-level dataraces, bringing the programmer's attention to pairs of transactions that the programmer has misspecified, and should have been combined into a single transaction. Our framework was validated against a set of programs with well known anomalies and demonstrated high accuracy and effectiveness, thus contributing for improving the correctness of TM programs.

Keywords: Static Analysis, Testing, Verification, Concurrency, Software Transactional Memory

1 Introduction

Concurrent programming is inherently hard. The fact that more than one ordering of events may take place at runtime makes it difficult to consider all possible execution scenarios of a program, and may expose unpredicted and harmful behaviors. The most notorious of these concurrency errors is the *datarace*, or *low-level datarace*, which happens when two threads concurrently access a shared variable with no concurrency control enforced, and at least one of those accesses is an update. Low-level dataraces may be avoided by using locks, thus establishing a mutual exclusion between certain code blocks of the program.

* This work was partially supported by Sun Microsystems and Sun Microsystems Portugal under the "Sun Worldwide Marketing Loaner Agreement #11497", by the Centro de Informática e Tecnologias da Informação (CITI), and by the Fundação para a Ciência e Tecnologia (FCT/MCTES) in the Byzantium research project PTDC/EIA/74325/2006 and research grant SFRH/BD/41765/2007.

Even in the absence of low-level dataraces, a program may still exhibit concurrency errors which result from incorrect ordering of correctly synchronized critical sections. Such is the case when a thread executes two separate critical sections which are related and should be merged into a single one in order to ensure correctness. We shall call these errors *high-level dataraces* or *high-level anomalies*. In contrast, dataraces, or low-level dataraces, will also be referred in this paper as *low-level anomalies*.

This paper addresses the detection of both low-level and high-level anomalies in the Transactional Memory (TM) [11,17] setting. TM is a promising approach that offers multiple advantages for concurrent programming. In contrast to the usage of locks, which enforces mutual exclusion, TM is neutral concerning the execution model. Memory transactions usually execute optimistically in real concurrency, assuming that no transaction will interfere with another. An underlying TM framework monitors the system and aborts conflicting transactions.

TM is inherently immune to some concurrency errors that storm lock-based programs, such as deadlocks. However, low-level dataraces can still be observed. A transaction is only shielded against another transaction, in the same way that a lock-protected critical section is only protected from another critical section which holds a common lock.

There are several approaches to detect dataraces in lock-based programs, both static [6, 9, 13], dynamic [8, 12, 15], and hybrid [16]. Likewise, there are many approaches for high-level anomaly detection [3, 5, 10, 20, 22]. Even though some results could also be applied to TM programs, none of these works targets specifically the TM setting. In this paper, we consider the different nature of TM programs, providing detection approaches for both low-level and high-level anomalies in TM. For this, we discuss how to apply a low-level anomaly detector meant for locks to a TM-based program, and we also propose a new definition for high-level anomalies. We also describe a new static approach for detecting high-level anomalies in TM, which conservatively extracts all possible execution traces of a program and searches for anomalies using a pattern-based approach.

We will discuss the detection of low-level TM anomalies in Sect. 2, and of high-level anomalies in Sect. 3; followed by a discussion of the related work in Sect. 4; and by the conclusions and future work in the last section.

2 Low-Level Dataraces

The TM approach provides several advantages over currently existing concurrency control mechanisms. In particular, the usage of TM alone guarantees the absence of some concurrency errors, such as deadlocks and priority inversion. However, depending on the particular underlying TM system, *dataraces* may still be observed. In this section we show how to detect dataraces in TM by converting transactions into lock-protected critical sections and applying an existing lock-oriented datarace detector.

2.1 Dataraces in Transactional Memory vs. Locks

Locks enforce mutual exclusion between critical sections. If two distinct critical regions are protected by at least one common lock, then no two threads may execute them at the same time. One the other hand, in most cases TM does not enforce mutual exclusion. Instead, two transactional code blocks may execute concurrently, provided with the guarantees of *Isolation* and *Atomicity*. TM provides *serializability* of transactions, ensuring that if two transactions take place concurrently and both succeed, then its final outcome is the same as if those two transactions were executed one after the other.

Consider the distinct situations that may lead to a low-level datarace between two threads, when using locks to control access to shared data:

1. None of the accesses is performed while holding a lock;
2. Only one of the accesses is performed while holding a lock; and
3. Both accesses are performed while holding locks; but there is no common lock shared between them.

With locks, the user chooses which critical section will be mutually exclusive with each other. Because in TM all transactions have the guarantees of Isolation and Atomicity against *all* other concurrent transactions, the third case above does not apply. Hence, as illustrated in Fig. 1, the situations that may lead to dataraces in TM are:

1. None of the accesses is performed in the scope of a transaction; or
2. Only one of the accesses is performed in the scope of a transaction.

The general idea of our approach is to use an existing datarace detection framework, interpreting atomic blocks as though they were simple lock-based synchronized blocks. We propose an approach to automatically convert transactional blocks into lock-based blocks, all synchronized on a single global lock. Fig. 1 illustrates how each datarace case in TM is to converted to a related situation with locks that result in the same outcome.

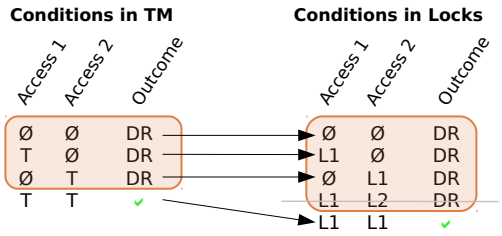


Fig. 1. Conditions for a datarace in TM and Locks

2.2 Detection Approach

Our approach was implemented through the use of AJEX [7], an extension to the Polyglot compiler framework [14] for Java, that already parses atomic blocks. Although our approach is independent from the lock-based datarace detector, in the current implementation we opted to use JChord [13].

The automatic transformation process of TM programs to synchronized single-lock programs consists of three distinct phases:

1. *Parse the source code with the AJEX Polyglot extension.* AJEX already handles atomic blocks, inserting them in the generated program's AST.
2. *Global lock generation.* The global lock must be globally accessible and have an unique, unused name. Our automatic approach automatically decides on how it should be named (a randomly generated name guaranteed to not collide with other already existing names); and where it should be declared (usually in the main class).
3. *Generation of the transformed program.* This new program contains synchronized blocks instead of the original atomic blocks. The datarace detector, JChord, is then invoked on the generated program and the results are presented.

2.3 Experiments

In order to validate our approach for transforming TM programs to synchronized single-lock programs, a set of validation tests have been carried out [19]. Some of these tests are well-known erroneous programs intended to benchmark validation tools like our own. Others were developed specifically to test our tool, containing simple stub programs with dataraces. We also tested Lee-TM [2], a renowned concurrency benchmark. For each test it was necessary to have both lock-based and TM-based versions, and some of the existing tests meant for locks had to be manually rewritten using TM.

Tests were carried out by initially running JChord on the lock-based version of each test. Then, we applied our approach to the TM versions of those tests, by transforming them into single global-lock programs and feeding them to JChord. For each test, the results for both executions of JChord were then compared. All the results obtained fit into one of the following scenarios: for tests where the lock-based and TM-based versions were strictly equivalent, the analysis results were equivalent as well; when the TM and lock-based versions of a test would have slightly different semantics (since some lock-based situations could not be modeled using the TM programming model), results were slightly different, but all those differences could be clearly mapped to the semantic variations between the two versions.

2.4 Discussion

We have presented a static approach to detect low-level dataraces in TM programs. This approach is carried out by automatically converting transactions

into proper lock-protected sections, and then invoking a lock-based datarace detector. We have elaborated on the experimental results that show the validity of this solution. The results have demonstrated that it is possible to detect real anomalies in TM programs with our approach. More details on the detection of low-level dataraces in TM programs can be found in [19].

3 High-Level Anomalies

A program that is free of low-level dataraces may still exhibit concurrency errors. Unlike low-level anomalies, high-level anomalies do not result from unsynchronized accesses to shared variables, but rather from a combination of multiple synchronized accesses, which may lead to incorrect behaviors if ran in a specific order.

As an example, consider the program in Fig. 2, showing a bounded data structure whose size should not go beyond `MAX_SIZE`. Before a thread asks for an item to be stored, it checks for available room. All accesses to the `list` fields are safely enclosed inside transactions, and therefore no low-level datarace may exist. However, due to interleaving with another thread running the same code, between the executions of `hasSpaceLeft()` and `store()` the size of the list could have changed to the maximum allowed; thus, the first thread would now be adding an element to an already full list. Therefore, both method calls should have been done inside the same transaction.

```
private boolean hasSpaceLeft() {
    atomic { return (this.list.size() < MAX_SIZE); }
}

private synchronized void store(Object obj) {
    atomic { this.list.add(obj); }
}

public void attemptToStore(Object obj) {
    if (this.hasSpaceLeft()) {
        // list may become full!
    this.store(obj);
    }
}
```

Fig. 2. Example of a High-level Anomaly

In the following sections we will discuss the conditions that may trigger high-level anomalies, propose a possible categorization of those anomalies, and present our approach for their detection.

3.1 Thread Atomicity

High-level anomalies are related with sets of transactions involving different threads, which leave the program in an inconsistent state if ran in a specific

order. This happens because two or more transactions executed by a thread are somehow related and make assumptions about each other (e.g., assumption of success), but there is a scheduling in which another thread issues a concurrent transaction which breaks that assumption. The simplest way to solve this problem is to merge those related atomic sections into a single transaction. Furthermore, through empirical observation, it seems that most or all of such anomalies involve only three transactions. Two consecutive transactions from one thread and a third transaction from another thread, that when scheduled to run between the other two, causes an anomaly.

Without further information from the developer on the intended program semantics, it is not possible to infer at compile time all the relations among transactions. However, it is possible to identify transactions that may or will affect other transactions, and use this information to identify possible high-level anomalies.

Consider a coordinate pair object shared between multiple threads. Let us assume that a thread T_1 issues a transaction $t_{1.1}$ to read value x, and then issues transaction $t_{1.2}$ to read y. In between them, thread T_2 could issue transaction $t_{2.1}$ which sets both values to 0, and so thread T_1 would have read values corresponding to the old x and new y (zero), when it is likely that both read operations were meant to read one single instant, i.e., either both before or after $t_{2.1}$.

In this scenario, the final outcome is not equivalent to a situation in which both read operations were ran without interleaving. The property of a set of threads whose interleavings are guaranteed to be equivalent to their sequential execution is called *thread atomicity* [22], and will be addressed again in Section 4. It is common to pursue thread atomicity as being a correctness criterion.

3.2 Anomaly Patterns in Transactional Memory Programs

Feeling that full thread atomicity is too restrictive, thus triggering many false positive scenarios, we opted for a more relaxed semantic that allows a restricted number of atomicity violations. As an example of an atomicity violation which in principle is not an error, consider Fig. 3, where each rectangle corresponds to a transactional code block. The second (read) operation in T_1 will be retrieving the results written by T_2. In order for this set of threads to be serializable, and thus thread atomic, all possible interleavings would have to be equivalent to the scenario in which the read immediately follows the write of the same thread.

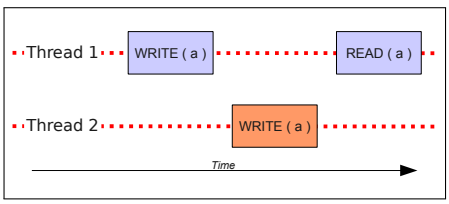


Fig. 3. An unserializable pattern which does not appear to be anomalous.

However, given the specific context of TM and the set of operations presented in the figure, it seems unintuitive that this particular set would contain an error. The read operation is retrieving a, and it seems unlikely that an operation will be performed based on the value written before by the same thread, as it would possibly be already outdated. The only error scenario involving this particular setup would be the case in which after the read, the first thread would do a set of operations that depend on both, the value just read and the value previously written (e.g., assuming them to be equal).

Therefore, we propose a framework for detecting a configurable set of patterns, and we opted to include only those most likely will result in concurrency anomalies. Out of all the patterns that incur in atomicity violations, we have isolated three suspicious patterns which describe possible high-level anomalies.

Read–write–Read or *RwR* — Non-atomic global read. A thread reads a global state in two or more separate transactions. If it make assumptions based on that state, this will most probably be a high-level anomaly.

Write–read–Write or *WrW* — Non-atomic global write. This is the opposite scenario from above. A thread is changing the global shared state, but in multiple separate transactions. Other thread reading the global state will observe this state as inconsistent.

Read–write–Write or *RwW* — Non-atomic *compare-and-swap*. A thread checks a variable value and, based on that value, alters the state of variable. If the variable is changed meanwhile, the update will probably not make sense anymore.

In the following, we will present our approach for statically matching suspicious patterns against the program source code, and will report on the experiments that assess the applicability and effectiveness of these patterns.

3.3 Symbolic Execution of Transactional Memory Programs

To detect high-level anomalies in TM programs, we perform a symbolic execution of the program and generate a set of possible execution traces of the transactional code. From these traces, we generate the set of possible interleavings of transactional code blocks and check if there are matches with any of the patterns identified in Sect. 3.2. Our approach for the detection of high-level anomalies in TM programs was also implemented using Polyglot framework and AJEX. As described before, Polyglot is a framework for performing transformations and analysis on Java programs, and AJEX is an extension to Polyglot that parses atomic blocks in TM programs.

The thread traces are obtained by symbolic execution. When the program to be analyzed is loaded, all class declarations that contain main thread methods are retrieved. This includes the classes that have a `public static void main (String args[])`, the classes that inherit `java.lang.Thread` or `java.lang.Runnable` and that contain a `run()` method declaration. Hence, we obtain a list of all thread bootstrap methods. Statements in these thread

methods are then analyzed. Whenever a transactional code block is found, it is added to the current trace, together with the full list of read and write operations of that transaction.

Challenges arise when the program code is not strictly linear. When a method call is encountered, the solution is to in-line the called code, i.e., to replace the method call with the body of the target method, so that the transactions performed by that method are still seen as being performed by the current thread. Care must be taken not to perform infinite in-linings when in the presence of recursive methods.

Additional challenges derive from disjunctions in the program control flow. When there are multiple alternative paths, such as when using *if* or *case* statements, the current trace must still represent all possible alternative paths. Rather than having numerous alternative traces for the same thread, our approach adds a special disjunction node to the trace, symbolizing a disjunction point, where the execution can follow one of the multiple alternative paths. Thus, the trace actually takes the form of a tree representing all the transactional blocks in all the possible execution paths for a thread.

Finally, we also have to deal with loop structures in the input program. This is solved by considering the representative scenarios of the execution of loops. Therefore, the trace tree considers the cases in which the loop 1) is not executed, 2) is executed once, or 3) is executed twice. It is not necessary to consider more than two consecutive executions, as all the anomalies detected with three or more expansions of the loop body are duplicates of those detected with just two expansions. On the other hand, two expansions of the loop body may yield anomalies that would not be detected with a single expansion. This is the case when the loop body includes two or more transactions that are involved in anomalies among themselves. Also, it is necessary to consider zero executions of the loop body, for the case in which the statements that precede and the ones that follow the loop are both involved in an anomaly.

3.4 Validation of the Approach

We ran a total of 14 tests, consisting of small programs taken from the literature [3, 4, 5, 10, 21], with studied high-level anomalies and that were analyzed by our tool. Additionally, one test consisted on the *Allocate Vector* from the IBM concurrency benchmark repository [1], and another test was developed by ourselves [19]. The results are summarized in Table 1.

In a total of 12 anomalies present in these programs, 10 were correctly pointed out: 2 *RwR* anomalies, 3 *WrW* and 5 *RwW*.

The false negatives were not due to imprecision of the anomaly patterns, but rather to data accesses in JRE classes, whose source code is not available. Those JRE methods may possibly read or update internal data, but since their code is not available, these methods are ignored, thus missing potential anomalies. As a possible approach to solve this problem, these unavailable methods could be assumed to read and modify the involved objects.

Table 1. Test results summary.

Test Name	Total Anomalies	Total Warnings	Correct Warnings	False Warnings	Missed Anomalies
Connection [5]	2	2	1	1	1
Coordinates'03 [3]	1	4	1	3	0
Local Variable [3]	1	2	1	1	0
NASA [3]	1	1	1	0	0
Coordinates'04 [4]	1	4	1	3	0
Buffer [4]	0	7	0	7	0
Double-Check [4]	0	2	0	2	0
StringBuffer [10]	1	0	0	0	1
Account [21]	1	1	1	0	0
Jigsaw [21]	1	2	1	1	0
Over-reporting [21]	0	2	0	2	0
Under-reporting [21]	1	1	1	0	0
Allocate Vector [1]	1	2	1	1	0
Knight Moves [19]	1	3	1	2	0
Total	**12**	**33**	**10**	**23**	**2**

In addition to the correctly detected anomalies, there were also 23 false positives (70% of total warnings). We group the causes for these imprecisions in 4 different categories.

First, out of these 23 false warnings, 5 were due to redundant reading operations. In a read operation `object.field`, two readings are actually being performed: `object` and `field`. It makes no sense that two instances of this statement be involved in a *RwR*. It would be possible to eliminate these false positives if the accesses were considered the same.

Another 6 false positives are related to cases for which additional semantic information would have to be provided or inferred. These false warnings could be eliminated with the aid of other available techniques, such as pointer analysis.

Of the remaining false positives, 10 could be eliminated by refining the definition of the *anomaly patterns* described in Section 3.2, with alterations that are indeed intuitive. For example, an *RwR* anomaly could be ignored if the second transaction would write both values involved. However, these alterations should be made carefully, as they could harm the overall behavior in other tests.

Finally, the remaining 2 false reports are also related to correct accesses which are matched by our anomaly patterns. Further study would be necessary to adapt the anomaly patterns in order to leave out these correct accesses, without seriously compromising the precision of the approach.

3.5 Discussion

We have analyzed common criteria for reporting high-level anomalies, and attempted to provide a more useful criteria by defining three anomaly patterns. We

have defined and implemented a new approach to static detection of high-level concurrency anomalies in Transactional Memory programs. This new approach works by conservatively tracing transactions and matching the interference between each consecutive pair of transactions against a set of defined anomaly patterns. Our approach raises false positives, although at an acceptable level. When compared with the existing reports from the literature, these results are somewhat better. The two false negatives were related to the unavailability of the source code and not to the inadequacy of the anomaly patters. We may therefore conclude that our conservative tracing of transactions is a reasonable indicator of the behavior of a program, since our results rival with those of dynamic approaches. More details on the detection of high-level dataraces in TM programs can be found in [18, 19].

4 Related Work

Low-level datarace detection, either by observing a program's execution — dynamic approach — or its specification — static approach — has been an area of intense research [6, 8, 9, 12, 13, 15, 16]. We are unaware of any work that specifically targets TM; however, we have shown that current algorithms, which are mainly intended for use with lock-based mechanisms, may as well be applied to transformed TM programs.

Although high-level anomaly detection is not such a hot topic, there are some relevant works which share some principles and features with our own. One of the earliest works on the subject is the one by Wang and Stoller [22]. They introduce the concept of thread atomicity, *atomicity* having a different meaning than the one stated in the ACID properties provided by TM systems. In this case, thread atomicity is more related to *serializability*, and it means that any concurrent execution of a set of threads must be equivalent to some sequential execution of the same set of threads. Wang and Stoller provide two algorithms for dynamically (i.e., at runtime) finding atomicity violations. Other authors have based on this work to develop other approaches [5, 10].

An attempt to provide a more accurate definition of anomalies is the work on *High-Level Dataraces* (HLDRs) by Artho et. al [3]. Informally, an HLDR refers to variables that are related and should be accessed together, but there is some thread that does not access that variable set atomically. This is different from thread atomicity, which considers the interaction between transactions, without regard for relations between variables.

Because HLDR is concerned with sets of related variables, some atomicity violations are not regarded as anomalies, such as those which concern only one variable. On the other hand, it is possible that an HLDR does not incur in an atomicity violation. This work is in some way related to ours, in that it attempts to increase the precision of thread atomicity by lowering its false positives. However, while our approach is to simply disregard some atomicity violations as safe, the work by Artho founds a new definition, which still exhibits some false positives, and also introduces some false negatives. This work is also related to

our in that they both automatically infer data relationships, and do not require processing user annotations which state those relationships.

A different approach has been taken by Vaziri et. al [20]. Their work focuses on a static pattern matching approach. The patterns reflect each of all the possible situations that may lead to an atomicity violation. The anomalies are detected based on sets of variables that should be handled as a whole. To this end, the user must explicitly declare the sets of values that are related. This work is similar to ours in that both approaches are static, and both follow a pattern-matching scheme. However, our approach is intended to be applied to existing programs, and so it assumes that any set of variables may be related. Contrarily, the work by Vaziri demands that the user explicitly declares which sets of variables are meant to be treated atomically, and so it can trigger anomalies on all atomicity violations, without too many false positives.

5 Concluding Remarks

We have proposed a framework for the detection of both low-level and high-level anomalies in Transactional Memory programs. The framework resorts to statical analysis of the program's source code to detect and report those anomalies.

The methodology used to detect low-level dataraces, based in a source-to-source transformation of a TM program to a lock-based one, was proven to provide adequate results, thus being a possible strategy to detect this kind of anomalies.

The methodology used to detect high-level dataraces, based in static analysis and symbolic code execution, and matching transactions' interleavings with suspicious patterns, has also provided good quality results, comparable to or even better than those reported in the literature for analogous problems in lock-based programs.

Our approach is novel because it is based in static analysis; it extracts conservative trace trees aiming at reducing the number of states to be analyzed; and it detects anomalies using a heuristic based in a set of suspicious patterns believed to be anomalous.

The developed tool could be improved by further refining the error patterns. The addition of *points-to* and *happens-in-parallel* analyses would also help to improve the tool by reducing the number of states to be analyzed. Other improvements could be achieved by enabling the analysis of standard or unavailable methods, and by solving the issue of redundant read accesses, as discussed in Sect. 3.4.

References

1. IBM's Concurrency Testing Repository. `https://qp.research.ibm.com/concurrency_testing`.
2. Mohammad Ansari et al. Lee-TM: A non-trivial benchmark suite for transactional memory. In *Proceedings of ICA3PP '08*, pages 196–207, Berlin, 2008. Springer-Verlag.

3. Cyrille Artho, Klaus Havelund, and Armin Biere. High-level data races. *Softw. Test., Verif. Reliab.*, 13(4):207–227, 2003.
4. Cyrille Artho, Klaus Havelund, and Armin Biere. Using block-local atomicity to detect stale-value concurrency errors. In Farn Wang, editor, *ATVA*, volume 3299 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2004.
5. Nels E. Beckman, Kevin Bierhoff, and Jonathan Aldrich. Verifying correct usage of atomic blocks and typestate. *SIGPLAN Not.*, 43(10):227–244, 2008.
6. Jong deok Choi, Alexey Loginov, Vivek Sarkar, and Alexey Logthor. Static datarace analysis for multithreaded object-oriented programs. Technical report, IBM Research Division, Thomas J. Watson Research Centre, 2001.
7. Ricardo Dias and Bruno Teixeira. Ajex: A source-to-source java stm framework compiler. Technical report, DI-FCT/UNL, April 2009.
8. Anne Dinning and Edith Schonberg. Detecting access anomalies in programs with critical sections. *SIGPLAN Not.*, 26(12):85–96, 1991.
9. Cormac Flanagan and Stephen N. Freund. Type-based race detection for Java. *SIGPLAN Not.*, 35(5):219–232, 2000.
10. Cormac Flanagan and Stephen N Freund. Atomizer: a dynamic atomicity checker for multithreaded programs. In *Proceedings of POPL'04*, pages 256–267, New York, NY, USA, 2004. ACM.
11. Maurice Herlihy, Victor Luchangco, Mark Moir, and III William N. Scherer. Software transactional memory for dynamic-sized data structures. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 92–101, New York, NY, USA, 2003. ACM.
12. Daniel Marino, Madanlal Musuvathi, and Satish Narayanasamy. Literace: effective sampling for lightweight data-race detection. In *Proceedings of PLDI'09*, pages 134–143, New York, NY, USA, 2009. ACM.
13. Mayur Naik, Alex Aiken, and John Whaley. Effective static race detection for java. In *PLDI*, pages 308–319. ACM Press, 2006.
14. Nathaniel Nystrom, Michael R. Clarkson, and Andrew C. Myers. Polyglot: An extensible compiler framework for java. In *CC*, pages 138–152, 2003.
15. Robert O'Callahan and Jong-Deok Choi. Hybrid dynamic data race detection. *SIGPLAN Not.*, 38(10):167–178, 2003.
16. Yao Qi, Raja Das, Zhi Da Luo, and Martin Trotter. Multicoresdk: a practical and efficient data race detector for real-world applications. In *Proceedings of the 7th Workshop on Parallel and Distributed Systems*, pages 1–11, New York, NY, USA, 2009. ACM.
17. Nir Shavit and Dan Touitou. Software transactional memory. In *Proceedings of PODC'95*, pages 204–213, New York, NY, USA, 1995. ACM.
18. B. Teixeira, D. Sousa, J. Lourenço, R. Dias, and E. Farchi. Detection of transactional memory anomalies using static analysis. In *Proceedings of PADTAD'10*, pages 26–36, New York, NY, USA, 2010. ACM.
19. Bruno Teixeira. Static detection of anomalies in transactional memory programs. Master's thesis, Universidade Nova de Lisboa, April 2010.
20. Mandana Vaziri, Frank Tip, and Julian Dolby. Associating synchronization constraints with data in an object-oriented language. In *Proceedings of POPL'06*, pages 334–345, New York, NY, USA, 2006. ACM.
21. Christoph von Praun and Thomas R. Gross. Static detection of atomicity violations in object-oriented programs. In *Journal of Object Technology*, page 2004, 2003.
22. Liqiang Wang and Scott D. Stoller. Run-time analysis for atomicity. *Electronic Notes in Theoretical Computer Science*, 89(2):191–209, 2003. RV '2003, Run-time Verification (Satellite Workshop of CAV '03).

Animation of Tile-Based Games Automatically Derived from Simulation Specifications

Jan Wolter, Bastian Cramer, and Uwe Kastens

University of Paderborn
Department of Computer Science
Fürstenallee 11, 33102 Paderborn, Germany
jwolter@mail.uni-paderborn.de,{bcramer,uwe}@uni-paderborn.de

Abstract. Visual Languages (VLs) are beneficial particularly for domain-specific applications, since they can support ease of understanding by visual metaphors. If such a language has an execution semantics, comprehension of program execution may be supported by direct visualization. This closes the gap between program depiction and execution. To rapidly develop a VL with execution semantics a generator framework is needed which incorporates the complex knowledge of simulating and animating a VL on a high specification level.
In this paper we show how a fully playable tile-based game is specified with our generator framework DEViL. We illustrate this on the famous Pac-man[1] game.
We claim that our simulation and animation approach is suitable for the rapid development process. We show that the simulation of a VL is easily reached even in complex scenarios and that the automatically generated animation is mostly adequate, even for other kinds of VLs like diagrammatic, iconic or graph based ones.

1 Introduction

A prominent representative of a visual language is the Unified Modeling Language (UML) [9] which is often used in software engineering process. Even smaller languages precoined for a specific domain are popular, because they can use visual metaphors of the target domain. In general an instance of such a visual language is used to produce source code of a different domain, e.g. Java Code from an UML class diagram.

To gain acceptance in rapid prototyping generator frameworks are used which can generate graphical structure editors for such visual languages from high-level specifications. These generators incorporate expert knowledge to produce a complete development environment for a VL with all features known from typical text editors like cut and paste, printing, drag and drop and so on. Unfortunately there is still a gap between program depiction and the generated code of that program. The programmer has to keep in mind what the program, he just created, does when it is executed. This gap is known as the gulf of execution [8]. Simulation and animation of the visual language instance can help to narrow

[1] Pac-man® is a registered trademark of Namco.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 113–124

this gap. The execution semantics of a visual language (if it has one) can be integrated into the visual language. Hence the VL instance is no longer static it can be simulated and smoothly animated. The user can "see" his language being executed before he generates code.

This helps to avoid mistakes at a very early stage and it supports program comprehension which is a challenging task especially in languages where many things happen in parallel.

The *Development Environment for Visual Languages*, DEViL, is a generator framework for visual languages which produces graphical editors from declarative high-level specifications. We extended it with simulation and animation support for VLs whereas a smooth and challenging animation can be derived automatically from a simple simulation specification. In this paper we want to show that our simulation specification language is powerful to simulate even complex behavior. We claim that the language helps in rapid prototyping, because simulation becomes an easy task due to powerful encapsulated concepts like event driven simulation and the extension of the simulation model. We will show that the automatically derived animation is suitable in most situations.

We will demonstrate this on the famous Pac-man game. It has a playful character, but it is also a challenging language for simulation, because of the complex navigation concepts of the "ghost" pawns in the game.

The paper is structured as follows: First we introduce the DEViL system and its underlying specification concepts with particular attention to simulation and animation. In Section 3 we give a brief description of the Pac-man game. In the next section we present our Pac-man Editor with special attention to the strategies of the ghost characters. Section 5 addresses related work and section 6 completes the exposition with a conclusion and a look at other implemented languages.

2 The DEViL System

The DEViL framework generates syntax-directed graphical structure editors for visual languages. The generated environments support all features of commonly used editors. Especially 2.5D views on the underlying semantic model are supported. A more in depth look at the generator framework and its generated products with respect to usability can be found in [12].

DEViL has already been successfully used for projects with nameable companies like Bosch [3], VW or SagemOrga [14]. The specification of this Pac-man Game Editor was one of many bachelor resp. master-theses that used the DEViL framework.

The specification process to generate "static" environments - environments without simulation and animation support - is divided into three parts. As can be seen later in this paper, simulation and animation support can be extended easily by the reuse of components of some of these three specification steps. Hence an user of the DEViL system who can build visual development environments can extend a language with simulation and animation support with reasonable effort.

To generate a structure editor for DEViL one first specifies the semantic model of the visual language. This is done with DSSL (DEViL Structure Specification Language). The semantic model abstracts from the visual representation. It stores just the information necessary to describe the semantics of the visual program. DSSL is inspired by an object oriented design with classes, inheritance, attributes and references. Fig. 1 shows a part of the specification of the semantic model for our Pac-man Editor. DEViL can generate an editor with a tree based structure manipulation view from this part of the specification.

```
CLASS Tile {
  columnRef: REF Column;
  item: SUB Item?;
}
ABSTRACT CLASS Item {
  name: VAL VLString;
}
CLASS Pacman INHERITS Item {
  direction: VAL VLInt INIT "2";
  angle: VAL VLInt INIT "0";
  clockwise: VAL VLBoolean;
}
```

Fig. 1. Part of the semantic model for the Pac-man Editor.

To obtain an advanced visual representation the semantic model (created with DSSL) is decorated with so called "visual patterns". Visual patterns define how constructs of the structure tree should look like. E.g. one can specify that some part of the structure tree should be laid out as the abstract concept "list" and aggregated nodes play the role of "list elements". Control attributes may modify this layout, for instance the list could be constituted vertically instead of horizontally. DEViL provides a huge library of precoined visual patterns with various possibilities to adapt their layout and appearance. A subset of this library is for example "sets, lists, trees, formulae or matrices". Technically, symbols of the semantic structure definition inherit from these visual patterns. The attribute evaluator generated by LIGA [4] of the underlying compiler generator framework Eli [5] computes the final graphical representation.

The last (optional) step of the specification process is the definition of a code generator. Here all of the tools of the Eli system to analyze the visual language instance can be used. A more detailed description of the VL specification process can be found in [13].

In order to separate concerns of specification simulation and animation are to be distinguished: simulation is the raw execution semantics of the visual language and animation is the smooth depiction of discrete execution of VL programs. Some visual languages have a precisely defined execution semantics, e.g. the firing of tokens in a Petri-net may be smoothly depicted by animation. For other visual languages simulation and animation may require to extend the semantic model to represent the simulation states or its graphical representation.

The presented Pac-man Editor (Fig. 4) considered as a visual language has a number of pawns that can be placed on a tile-based board which constitute the playing field. The pawns are typed structure objects of this VL. The Pac-man Editor has only four different pawns: "wall", "ghost", "powerpill" and "pac-man". Additionally, some structure objects are needed to represent the rows and columns of the board. Hence, our Pac-man VL is a playground editor where the user can create custom levels.

To specify a simulation for the Pac-man Editor we have to define the state space and the state transitions. Both can be specified in our simulation specification language DSIM.

Fig. 2 (a) shows the specification of the simulation model in DSIM. As can be seen, we again reuse DSSL concepts and we can extend the semantic model of the visual language to reach a new model that is suited for simulation. In this case we extended the semantic model class `Tile` (see Fig. 1). We can introduce new attributes that are needed for simulation purposes only or extend our simulation model with so called path expressions to traverse the simulation model tree at run time. Both model the state space for the simulation.

We could also narrow the semantic model of the visual language in our simulation model. This can be done if parts of the semantic model of the visual language are only needed for representation purposes and not for simulation.

```
MODEL {
  CLASS Tile {
    OBJECT pill OF PowerPill: "THIS.item.CHILDREN[0]";
    position: VAL VLPoint?;
    diffVal: VAL VLInt INIT "0";
    visited: VAL VLBoolean INIT "0";
  }
  CLASS Pacman {
    OBJECT tile OF Tile: "THIS.PARENT.PARENT";
  }
}
```

(a) Simulation model.

```
EVENTS {
  goGhost(Tile from, Tile to){
    Item i = REMOVE(from.item, FIRST);
    INSERT(to.item, i, FIRST);
  }
  eatPacman(Tile from, Tile to){
    IF(#[0]Root.sound == VLBoolean(1)){
      vlPlaySound("pacmanDeath.wav");
    }
    REMOVE(to.item, FIRST);
    Item i = REMOVE(from.item, FIRST);
    INSERT(to.item, i, FIRST);
    FIRE gameLost(#[0]Root)@TIME_NOW + 1;
  }
}
```

(b) Events.

Fig. 2. Simulation model in DSIM and some events which can be scheduled in the simulation loop.

```
FOREACH ghost IN [Ghost] {
  IF(ghost.strategy == VLInt(1)) {
    Tile to = NEIGHBOUR_TILE(mapping, NEUMANN, ghost.tile, Pacman);
    IF(NOTNULL(to) AND (ghost.eatable == VLBoolean(0))){
      FIRE eatPacman(ghost.tile, to) @TIME_DIRECT;
    }
    ELSE {
      IF(NOTNULL(#[0]Pacman)) {
        Tile to = NEIGHBOUR_EMPTY_TILE_RANDOM(mapping, NEUMANN, ghost.tile);
        IF(NOTNULL(to)) {
          FIRE goGhost(ghost.tile, to) @TIME_DIRECT;
        }
      }
    }
  }
}
```

Fig. 3. Part of the simulation loop.

Fig. 3 shows an excerpt of the behavior specification part of DSIM. Here the simulation model can be inspected and events can be scheduled that modify an instance of the simulation model. Hence we follow the event based approach to simulation. Events are scheduled for an arbitrary time. Any event can trigger arbitrary so called simulation modification actions. These actions modify the simulation model and they constitute the interface to the animation framework. The excerpt shows the behavior specification of the ghost pawns. They try to eat Pac-man if it is located on a neighbour tile. If not the ghost moves according to its strategy to the next tile.

In DSIM the following simulation modification actions exist which also form the interface to the animation part:

– `REMOVE` a structure object.

– `INSERT` a new structure object instance or insert a structure object, that was removed before. The latter would yield a `MOVE` action.
– `COPY` a structure object.
– `CHANGE_VAL` to change a primitive attribute or a reference.

In Fig. 2 (b) some events with corresponding simulation modification actions can be seen. The specific characteristics of the simulation modification actions is that an animation can automatically derived from such a specification.

The default animations triggered are: *slow shrinking* to invisibility of an object that is removed, *slow growing* of an object that is newly inserted. *Linear moving* (with optional easing) of a structurally moved object. Copied objects move from their copy source to their destination while changing their transparency value from invisible to visible. Since editors generated by DEViL are syntax directed structure editors, the creation or removal of structure objects can have side effects to other structure objects with respect to their size or position. These objects are automatically adapted smoothly, they are *morphed*. Even colors of structure objects are adapted smoothly.

The default animation behaviour is sufficient for most automatically derived animations as can be seen later. But, in some cases the default animation that is automatically triggered is not what the animator of a visual language desires. Here the animator can override the default behavior with so called animated visual patterns (AVPs). The AVPs can be decorated like the visual patterns to a structure object and tell the structure object in what way it is animated if a certain simulation modification action occurs. For instance if a token in a Petri-net is removed it should not shrink to invisibility which is the standard animation. The desired animation is to move the token to the fired transition, hence the used AVP to override the default behavior for remove is *AVPOnRemoveMove*. We have AVPs for changing size and transparency values of structure objects, for moving, scaling, rotating and so on. All of them can be combined and adapted to the needs of the animation.

A more detailed description of DEViL's simulation and animation facilities can be found in [2].

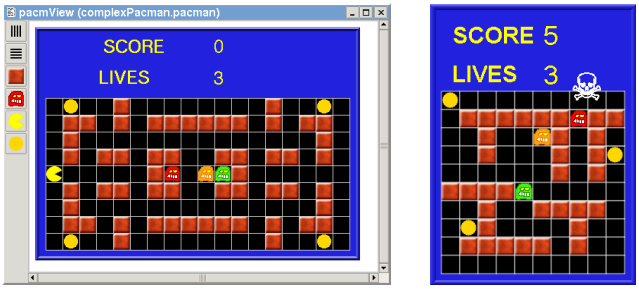
3 Pac-man

Pac-man is the most popular arcade computer game in the eighties of the last century and it was originally developed by Toru Iwatani for the Namco company in 1980. Because of the large degree of esteem different versions of the game have been reprogrammed many times for several systems like home computers, gameconsoles and recently even for the iPhone [7]. The game is very interesting in terms of navigating a character around a structured playground, accumulating points, avoiding and (in some cases) attacking non-player game characters.

The classic version of Pac-man is an one-player game where the human player routes the Pac-man around a maze with the goal to avoid the four *ghost* characters and to eat as much pills as possible. Initially the pills are placed in each walk-in field of the playground and will be eaten via the achievement of the field by Pac-man. The overall four ghosts roam through the maze trying to catch Pacman. This is successful when a ghost achieves a tile in which Pac-man is located.

In this case Pac-man looses one of his initial three lives and the game restarts when Pac-man has just one life. Each of the four ghosts pursues a different strategy to eat the Pac-man.

Besides the normal pills in each tile there are four *powerpills* which are located near each corner of a maze. When Pac-man eats a powerpill he gets a special score and is able to eat ghosts on his part. In this case all ghosts change their color to blue for few moments, reverse their direction, and usually move more slowly. If Pac-man eats a ghost, he gets a special score and the ghost resurrects in the middle of the maze after a few moments. In addition to the previously seen options there is one more possibility to increment the score: sometimes a symbol of a fruit appears at a random position of the maze, which also gives the chance to get extra points.



(a) Starting position of a Pac-man game.

(b) A dynamic animation object.

Fig. 4. Screenshots of our Pac-man game.

The game ends when all pills have been eaten or Pac-man has lost all of his lives. In the former case the player reaches a new level which is more difficult than the previous one. This can be achieved for example by faster moving ghosts.

Fig. 4 (a) shows a playground of a Pac-man game, which has been build with our Pac-man Editor. Besides the Pac-man the figure shows three different ghosts, wall items and powerpills.

4 Pac-man Editor

Our Pac-man Editor is structured as a *multi document interface* (MDI) and offers the ability to create user-defined playgrounds for Pac-man games. The user has the option to insert different items to the playground, e.g. Pac-man, ghosts, powerpills or wall items. It is also possible to expand the playground by adding rows and columns. A playground which is constructed in such a way allows to play Pac-man as mentioned above.

The specification of the semantic model was the first task to implement this editor. The most important part of the semantic model is the matrix structure. An object of the matrix class is associated with an arbitrary number of columns and rows. Each row owns several tiles, which includes in turn an item or not. The item is an abstract class and the concrete subclasses are either Wall-item, Pac-man, Ghost or Powerpill.

To realize a correct semantic playground it is essential to avoid more than one Pac-man or a game without Pac-man. Hence, the DEViL System provides

the ability to specify consistency constraints on various levels. E.g. cardinalities in the semantic model or specialized callback functions which can navigate the structure tree. All these checks are automatically performed before simulation. Hence, only a correct Pac-man game instance can be simulated.

Besides the consistency constraints the language designer can implement initialization functions for each class of the semantic structure. Such a function is a callback function and will be automatically called by the system, if a new object has been created. We used this for example to realize a default playground dimension of 10×10 tiles.

4.1 Strategies

With our editor it is possible to allot one of overall three different strategies to each ghost. We draw our inspiration with respect to the strategies of Repenning [11]:

Random The ghost roams randomly through the maze. At each step it evaluates the walk-in fields in the *von Neumann neighbourhood* and chooses one randomly.

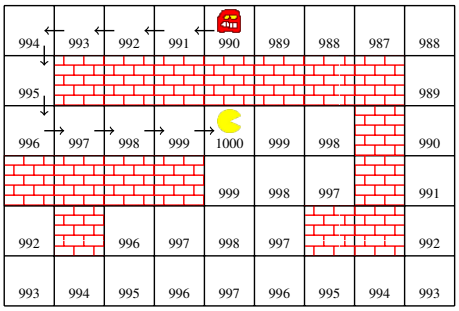


Fig. 5. Distribution of diffusion values to apply hill-climbing.

Incremental Approach The ghost tries to move closer to the Pac-man. At each step it evaluates the empty neighbour tiles and selects the closer closest one in euclidean sense.

Hill-climbing Due to the fact that the incremental approach does not permit the overcoming of walls, the strategy of hill-climbing affords this. To achieve this goal, diffusion values are used for each tile. These are used to spread the "scent" of the Pac-man in the maze. The value represents the closeness of a ghost to Pac-man. The largest value gets the tile in which Pac-man is allocated. Starting from this tile, the value is distributed to all walk-in fields of the playground. Every tile which is not accessible, e.g. a tile with a wall item, gets a negative diffusion value. At each step of the game the ghost selects the tile which has the largest diffusion value. Due to the fact that the diffusion values must be recalculated at each step, this brings the ghost closer to Pac-man. Fig. 5 illustrates the allocation of diffusion values and the way a ghost must go to get Pac-man.

A closer look to the implementation of the hill-climbing strategy is available in the next section. Amongst other things we describe the implementation of ghost strategies in DSIM.

4.2 Simulation

The user interaction via keyboard is essential for the Pac-man game. The DEViL System provides the ability to define arbitrary keyboard events which can be processed in the simulation.

We used the simulation model to add particular attributes which are necessarily needed for the simulation. An extract is given in Fig. 2 (a). We extend the Pac-man class with a tile attribute, which allows the access of the tile in which Pac-man is located, from the context of a Pac-man object. Besides others, we had extended the tile class with an attribute which stores the diffusion value of a tile. This is needed to realize the hill-climbing strategy. Keep in mind, that these attributes only exist in the simulation model, not in the semantic model of the Pac-man VL.

```
coordinatePacman(Pacman pacman, VLInt direction){
  Tile go = NEIGHBOUR_TILE(default, NEUMANN, pacman.tile, PowerPill);
  IF(NOTNULL(go)){
    FIRE eatPowerpill(pacman.tile, go, pacman, direction) @TIME_DIRECT;
  } ELSE {
    go = NEIGHBOUR_TILE(default, NEUMANN, pacman.tile, Ghost);
    IF(NOTNULL(go)){
      FIRE eatGhost(pacman.tile, go, pacman, direction) @TIME_DIRECT;
    }ELSE{
      go = NEIGHBOUR_TILE(default, pacman.tile, direction);
      IF(NOTNULL(go) AND (SIZE(go.item) == VLInt(0))){
        FIRE goPacman(pacman.tile, go, pacman, direction) @TIME_DIRECT;
      }
    }
  }
}

goPacman(Tile from, Tile to, Pacman p, VLInt d){
  FIRE incrementScore(#[0]Score, 1) @ TIME_DIRECT;
  FIRE computeRotation(p,d) @ TIME_DIRECT;
  Item i = REMOVE(from.item, FIRST);
  INSERT(to.item, i, FIRST);
}
```

Fig. 6. Coordination of the Pac-man pawn.

In the event block we specified events, which can be scheduled at an arbitrary simulation time in the loop block. Hence, our simulator follows an event driven approach. We had implemented overall 16 different events. Fig. 6 shows two events. The `coordinatePacman` event gets the Pac-man instance and a direction to move to. It checks, whether a powerpill or a ghost is in the way. If so Pac-man tries to eat the ghost resp. the powerpill. If there is nothing to eat Pac-man just walks to the next tile, the `goPacman` event is called. This event again calls two events to increment the score and to compute the rotation, which is needed for the animation. Finally the Pac-man pawn is removed from the actual tile and inserted to the tile in the desired direction. This yields a `MOVE` action for the Pac-man pawn.

```
NEIGHBOUR_COUNT(mapping, MOORE, pacman.tile, Ghost);
NEIGHBOUR_TILE(mapping, ghost.tile, S);
NEIGHBOUR_TILE_RANDOM(mapping, NEUMANN, pacman.tile);
```

Fig. 7. Exemplary neighbour access functions.

In each simulation step we have to compute the diffusion value for the hill-climbing strategy. This is done by a call of a C function. The function computes the value via a simple breadth-first search. Afterwards the ghost has to pick the target tile which has the largest diffusion value. To get a specific neighbour, we extended the simulation language such that we can access structure objects (of a specific type) in the neighbourhood of a given tile. Due to the fact that all editors generated with DEViL that make use of tiling have the same underlying

model we could identify a subset of tile-access functions which are often needed and generalize these functions. This lead to a decrease of hand written C-code.

Fig. 7 shows some neighbour access functions. The first function counts the ghosts in *Moore neighbourhood* of the Pac-man. A computation of the neighbour tile in south direction of a ghost shows the second function. The last function returns a random tile in *von Neumann* neighbourhood of Pac-man.

4.3 Animation

The default animation which is automatically derived from the simulation specification is almost adequate. A ghost and the Pac-man move fast from the start tile to the target tile in each simulation step. This is the case, because the animation framework interprets the modification actions `REMOVE` and `INSERT` as a *moving* animation. Furthermore the Pac-man shrinks to invisibility when he is caught by a ghost.

But Pac-man looks in the desired viewing direction until he has accessed the target tile. It would be nicer if Pac-man rotates to the desired viewing direction in the start tile before he moves to the target tile. Now the idea is to override the default behaviour for Pac-man. All animations are typed over their simulation modification action. Hence, we need to override the default animation pattern `MOVE`, because the Pac-man is moved (`REMOVE`d and `INSERT`ed) on the playground. We do it with the specification in Fig. 8 (a). We use the animated visual patterns `OnMoveRotate` and `OnMoveMove`. `OnMoveRotate` rotates a structure object if it is moved. Hence, we have to override the angle and rotate attributes. The angle and rotate attributes are stored in the "pacman" class (see Fig. 1) and will be computed via the *computeRotation* event in each simulation step. In addition we override the duration attribute to specify the duration of the rotate operation. In this configuration the rotation and the move are scheduled at the same simulation time, but we want the animation of the rotation to appear before the animation of the move. Hence the `OnMoveMove` animation must be animated after the `OnMoveRotate` animation. Hence, we have to assign the value 2 to the time attribute. Furthermore we override the duration attribute to indicate the duration time for a move operation. As can be seen, besides the simulation time, we have an animation time which defines an order of the animations and which can easily be adapted to gain a desired animation.

```
SYMBOL pacmView_Pacman INHERITS VPContainerElement, VPForm,
       AVPOnMoveRotate, AVPOnMoveMove, AVPOnRemoveShrink
COMPUTE
  SYNT.drawing = ADDROF(PacmanDrawing);
  SYNT.onMoveRotateAngle = THIS.pers_angle;
  SYNT.onMoveRotateClockwise = THIS.pers_clockwise;
  SYNT.onMoveRotateDuration = 600;
  SYNT.onMoveMoveRaiseDisplayOrder = 1;
  SYNT.onMoveMoveAnimationTime = 2;
  SYNT.onMoveMoveDuration = 900;
  SYNT.onRemoveShrinkAnimationTime = 10;
END;
```

(a) Mapping of AVPs with control attributes to override default animation.

```
SYMBOL pacmView_Pacman INHERITS AVPCreateDynamicObject,
    AVPMoveDynamicObject
COMPUTE
  SYNT.createDynamicObjectModificationAction = REMOVE;
  SYNT.createDynamicObjectDrawing = ADDROF(SkullDrawing);
  SYNT.createDynamicObjectPosition = POSITION(
          SELECT(THIS.oPosition, sub(VLPoint(0,10))));

  SYNT.moveDynamicObjectDuration = 8000;
  SYNT.moveDynamicObjectStartPosition = POSITION(
          SELECT(THIS.oPosition, sub(VLPoint(0,10))));
  SYNT.moveDynamicObjectEndPosition = POSITION(
          SELECT(THIS.oPosition, sub(VLPoint(0,60))));
END;
```

(b) Creating a dynamic animation object and moving it.

Fig. 8. Animation of Pac-man.

The animation framework offers the possibility to animate objects which are not part of the semantic model (so called *dynamic objects*). If Pac-man is caught,

we have used this feature to display a skull (see Fig. 4 (b)). For such a purpose it is only necessary to use the provided visual patterns as described in Fig. 8 (b). The visual pattern `CreateDynamicObject` reacts to a modification action and offers the possibility to add a drawing. As seen in Fig. 8 (b) we override the modification action attribute to react to a remove action. Furthermore, we override the drawing attribute to add the skull drawing. In order that the skull moves bottom-up from the position of the Pac-man, we had used the pattern `MoveDynamicObject`. We also had used the pattern `OnRemoveShrink` to show the skull temporary.

The specification of an animation in DEViL is straight forward: first specify a simulation, then derive the animation automatically. Hence the animation is a formal mapping of its simulation part. At last animations can be adapted by overriding the default animations through the application of a huge declarative animation pattern library.

5 Related Work

The Agentsheets system [10] can generate tile based simulations and games. The specification process is fully graphical and rule based. Agentsheets uses the programming by demonstration paradigm. In the rules one can access neighbour tiles through the help of icons with specific arrows. This is the visual variant of our neighbour access functions. Agentsheets is restricted to tile based simulation whereas our system can also handle diagrammatic or iconic visualizations.

In the area of generator frameworks for visual language environments the GenGed [1] system makes use of graph transformation and visual rewrite rules to specify simulation. To store the simulation state, rules must be extended. This is similar to our simulation model which can extend the semantic model of a VL. GenGed uses a formal mapping between simulation and animation. This is comparable to our simulation modification actions which trigger default animations. They are the interface between simulation and animation framework. Smooth and complex animations can not be specified with GenGed.

The DiaGen [6] system also uses graph transformation to specify a visual language. Some editors already support simulation and animation. Interesting is that every animation step is a state of the system whereas we interpolate between two adjacent simulation model states.

6 Conclusion

The specification of the Pac-man editor is a straight forward task. Table 1 shows that we needed 220 LOC for the whole simulation part including all ghost strategies and 400 LOC for hand written C-code. The other specs part needs only 236 LOC. The second column of the table shows a decrease of total LOC from 883 to 646 LOC. This is because of the neighbour access functions we had generalized. This reduced the LOC of C-code nearly by 250 LOC and we need only 7 additional LOC in the simulation specification to realize to mapping between concrete and generalize matrix structure. The 156 LOC of C-code is just a simple tile initialization function for the hill-climbing strategy. The automatically

derived animation is sufficient to play the game. The 27 LOC are just syntactic sugar.

The DSIM language with its narrow interface to the animation and its constructs tailored for the simulation of visual languages has already been approved in other VLs with execution semantics like Petri-nets, a Datapath simulation, electronic circuits and even the game Ludo. Table 2 shows the amount of LOC for simulation and animation part of already implemented editors. The examples in line two and three are based on the Petri-net simulation shown in line one. They show the simulation of the well-known dining philosophers and a simulation of a signal light of a four-way-crossing. Both are structural coupled to the Petri-net with DEViL's internal declarative coupling mechanism. A simulation of the Petri-net automatically triggers synchronization functions in the philosophers resp. signal-light view. The simulator detects these triggerings and calls the animation framework. Hence, additional specification amount is not needed.

	LOC	**LOC with access fct.**	**generated LOC**
simulation	220	227	87.504
animation	27	27	
C-code	400	156	
other specs.	236	236	
	883	646	87.504

Table 1. Distribution of the specification complexity.

	Simulation	**Coupling**	**Animation**	**Anim. syntactic sugar**
Petri-nets	29		4	0
Dining-Philosophers	29	95	4	0
Signal-Lights	29	57	4	0
Logo	211		3	3
Game of Life	39		0	0
Ludo	338		0	0
Statecharts	78		2	0
Bubblesort	13		0	0
Quicksort	93		0	0
CPU Datapath	263		160	0
Washing bay	35		0	0
Electronic circuits	99		109	0

Table 2. Simulation and Animation LOC of other VLs.

As can be seen in Table 2 the automatically triggered animation is mostly sufficient. We need to adapt the animation only in simulations where animations depend on the context of their structure objects. E.g. this is the case in our CPU datapath simulation where an animation of an instruction is different whether it is located in an instruction decoder, in an accumulator or somewhere else.

The already implemented VLs have a very diverse appearance: we have diagrammatic, iconic and graph based depictions. Currently we are working on a traffic simulation to see how our approach scales. Interesting extensions would be semantic zooming, camera views or even isometric views. Here also a pattern based approach is imaginable.

Also outstanding is a visual language for DSIM and an usability study.

References

1. Bardohl, R.: GenGed: A generic graphical editor for visual languages based on algebraic graph grammars. In: 1998 IEEE Symp. on Visual Lang. pp. 48–55 (Sep 1998)
2. Cramer, B., Kastens, U.: Animation automatically generated from simulation specifications. In: VLHCC '09: Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 157–164. IEEE Computer Society, Washington, DC, USA (2009)
3. Cramer, B., Klassen, D., Kastens, U.: Entwicklung und Evaluierung einer domnenspezifischen Sprache fr SPS-Schrittketten. In: Fahland, D., Sadilek, D.A., Scheidgen, M., Weileder, S. (eds.) DSML. CEUR Workshop Proceedings, vol. 324, pp. 59–73. CEUR-WS.org (2008), http://dblp.uni-trier.de/db/conf/dsml/dsml2008.html#CramerKK08
4. Kastens, U.: An attribute grammar system in a compiler construction environment. In: Proceedings of the International Summer School on Attribute Grammars, Application and Systems. Lecture Notes in Computer Science, vol. 545, pp. 380–400. Springer Verlag (1991)
5. Kastens, U., Pfahler, P., Jung, M.: The Eli system. In: Koskimies, K. (ed.) Proceedings of 7th International Conference on Compiler Construction CC'98. pp. 294–297. No. 1383 in Lecture Notes in Computer Science, Springer Verlag (Mar 1998)
6. Minas, M.: Concepts and realization of a diagram editor generator based on hypergraph transformation. Science of Computer Programming 44(2), 157–180 (Aug 2002), http://www.elsevier.com/gej-ng/10/39/21/86/49/29/abstract.html
7. Namco Games: Pacman for iPhone. http://www.appsafari.com/games/2741/pacman-for-iphone/ (2008), [Online; accessed 19-February-2010]
8. Norman, D.A., Draper, S.W.: User Centered System Design; New Perspectives on Human-Computer Interaction. L. Erlbaum Associates Inc., Hillsdale, NJ, USA (1986)
9. Object Management Group: Unified Modeling Language (UML), version 2.2 (2009), http://www.omg.org/technology/documents/formal/uml.htm
10. Repenning, A.: AgentSheets®: an Interactive Simulation Environment with End-User Programmable Agents. In: Interaction 2000, Tokyo, Japan (2000)
11. Repenning, A.: Collaborative Diffusion: Programming Antiobjects. In: OOPSLA 2006, ACM SIGPLAN International Conference on Object-Oriented Programming Systems, Languages, and Applications, (Portland, Oregon, 2006). IEEE Press (2006)
12. Schmidt, C., Cramer, B., Kastens, U.: Usability evaluation of a system for implementation of visual languages. In: Symposium on Visual Languages and Human-Centric Computing. pp. 231–238. IEEE Computer Society Press, Coeur d'Alne, Idaho, USA (Sep 2007)
13. Schmidt, C., Kastens, U., Cramer, B.: Using DEViL for implementation of domain-specific visual languages. In: Proceedings of the 1st Workshop on Domain-Specific Program Development. Nantes, France (Jul 2006), http://ag-kastens.upb.de/paper/dspd2006-devil.pdf
14. Schmidt, C., Pfahler, P., Kastens, U., Fischer, C., Gmbh, O.K.: Simtelligence designer/j: A visual language to specify sim toolkit applications. In: Proceedings of the Second Workshop on Domain Specific Visual Languages (OOPSLA 2002 (2002), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.9269

Domain-Specific Language for Coordination Patterns

Nuno Oliveira[1], Nuno Rodrigues[2], and Pedro Rangel Henriques[1]

[1] University of Minho - Department of Computer Science,
Campus de Gualtar, 4715-057, Braga, Portugal
{nunooliveira,lsb,prh}@di.uminho.pt
[2] IPCA – Polytechnic Institute of Cavado e Ave
Campus da ESGT, Barcelos, Portugal
nfr@ipca.pt

Abstract. The composition of software systems in a new one requires a good architectural design phase to speed up remote communications between these systems. But at the implementation phase, the code to coordinate such components ends up mixed in the main business code. This leads to maintenance problems, raising the need of, on the one hand, separate the coordination code from the business code, and on the other hand provide mechanisms for analysis and comprehension of the architectural decisions once made.

In this context our aim is at developing a domain-specific language, `CoordL`, to write coordination patterns. From our point of view, coordination patterns are abstractions, in a graph form, over the composition of coordination statements from the system code. These patterns would allow us to identify, by means of pattern-based graph search strategies, the code responsible for the coordination of the several components in a system. The recovering and separation of the architectural decisions for a better comprehension of the software is the main purpose of this pattern language.

1 Introduction

Software Architecture [12] is a discipline within the Software Development [11] concerned with the design of a system. It embodies the definition of a structure and the organization of components which will be part of the system. The architecture design concerns also with the way these components interact with each other and which are the constraints in their interactions. In their turn, software components [8] may be seen as objects in the object-oriented paradigm, however, besides data and behavior, they may embody whatever we think as a software abstraction. Although they may have their own functionality (sometimes a component is a remote system), most of the times they are developed to be composed with other components within a software system and to be reused from a system to another, giving birth to component-based software engineering methodology [9].

The definition of the interaction between the components of a system may be seen from two perspectives: (i) integration and (ii) coordination. The differences between these two perspectives is slightly none. The former is related with the integration of some

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 125–136

functionalities of a system into a second one which needs to borrow such a computation; the latter is concerned with the low level definition of the communication and its constraints between the modules of a system. Such interaction definition between the components should be *exogeneous*, that is, the coordination of components is made from the outside of a component, not needing to change its internals to make possible the communication with new components [3].

However, this *rule* of separating computational code from the coordination code is not always adopted by those who develop software. The code is all weaved in a single layer where there is no space for separation of concerns. This behavior would arise problems in the future of the system, namely in maintenance phases. These problems are mainly concerned with the comprehension of the code and architectural decisions by hampering their analysis.

Reverse engineering [20] of legacy systems for coordination layer recovery would play an important role on maintenance phases, diminishing the difficulties on analyzing the architectural decisions. But extracting code dedicated to the coordination of the system components from the whole intricate code is not an easy task. There is not a standard (and unique) way of programming the interactions between the components. However, and fortunately, there are a lot of code patterns which the majority of the developers use to write the coordination code. Once the code of a system can be represented as a graph of dependences between the statements and procedures, the so-called *System Dependence Graph* (SDG) [10], we are also able to represent code patterns as graphs, allowing the search for these patterns in the SDG.

In this context, we define the notion of *coordination patterns* as follows:

> Given a *dependence graph* $\mathcal{G}$ as in [18] a coordination pattern is an equivalence class, a *shape* or a sub-graph of $\mathcal{G}$, corresponding to a trace of coordination policies left in the system code.

In this paper we show how we developed a *Domain-Specific Language* (DSL) [22] named CoordL, to write coordination patterns. Our main objective is to translate CoordL specifications into a suitable graph representation. Such representation feed a graph-based search algorithm applied on a dependence graph, in order to discover the coordination code weaved in the system code. In more detail, the paper follows this structure: in Section 2 we address related work; in Section 3 we present and describe the syntax of CoordL; in Section 4 we address its semantics; in Section 5 we show how we used the AnTLR system to define the syntax and the semantics of CoordL; in Section 6 we expatiate upon actual and future applications of the language and the patterns and finally, in Section 7 we conclude the paper by expressing what we have done.

2 Related Work

CoordL is a DSL to write coordination patterns with the purpose of extracting and separating the coordination layer from the source code of a multi-component software

system. The main idea of this code separation into concern-oriented layers is to recover architectural decisions and ease the comprehension of the system and its architecture.

The recovery of the system architecture for software comprehension is not a novelty. Tools like Alborz [19] or Bauhaus [16] recover the blueprints of an object-oriented system. Bauhaus recovers architectures as a graph where the nodes may be types, routines, files or components of the system, and the edges model the relations between these nodes. Such architecture details are presented in different views for an easy understanding of the global architecture. Alborz presents the architecture as a graph of components and keeps a relation between this graph and the source code of the software system.

Our main aim is not at visualizing the blueprints of a system, but to provide mechanisms for understanding the rationale behind the architectural decisions once made. This embodies the recovering of the coordination code. The tools mentioned before do not support this feature and do not take advantage of code patterns to do the job.

Although we may reference Architectural Patterns [4] or Design Patterns [5] as related work, because of the common methodology of patterns and the borrowed notions and description topics, there is a huge difference between their application. While coordination patterns are used to lead a reverse and a re-engineering to recover architectural decisions, and are focused on low-level compositions of code, the architectural/design patterns play in a higher level, being used to define the architecture of a system in earlier phases of the software development process [11].

Architecture Description Language (ADL), are languages to formally describe the architectures and the interactions between the components of the system. Although CoordL is not to be considered an ADL, we must acknowledge that there are some similarities in the concepts embodied in these languages and those encapsulated in our. Some of these languages are ACME [7], ArchJava [1], Wright [2], and Rapide [13]. The great majority of these languages has tool support for analyzing the described architecture. Such analysis made at high level, allows one to reasoning about the correctness of the system, and may provide important information about future improvements that can or can not be done according to the actual state of the architecture.

According to our knowledge, there is no language with the same exact purpose as CoordL.

3 CoordL - Design and Syntax

The design of a DSL is always a task embodying a lot of steps. As a first step it is needed to collect all the information about the domain in which the language will actuate. Then this information shall be organized using, for instance, ontologies [21]. Once the main concepts of the domain are identified it is needed to choose those that are really needed to be encapsulated in the syntax of the language; this leads to the last step which concerns with the choice of a suitable syntax for the language.

Figure 1 presents an ontology to organize the domain knowledge of the area where we want to solve problems. The main concept of this domain is the coordination pattern. The majority of the concepts incorporated in this domain description are wider than

what we show, however, to keep the description limited to the domain, we narrowed the possible relations between each concept, as well as the examples they may have.

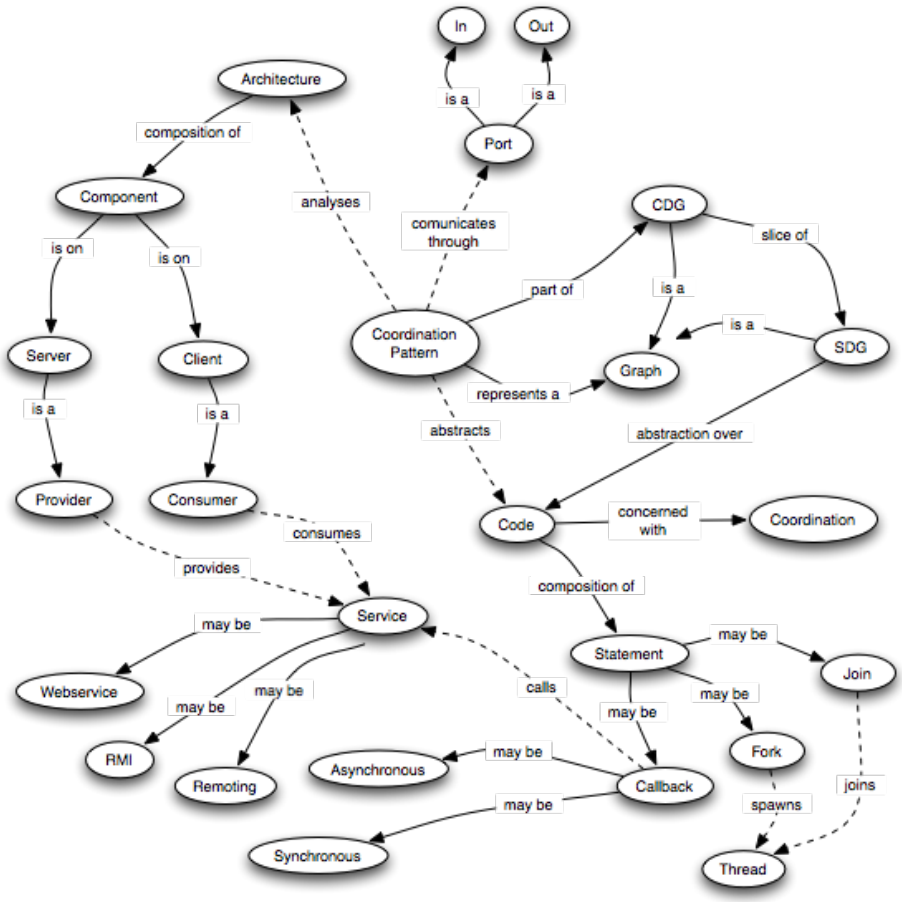


Fig. 1. Ontology Describing the Coordination Pattern Domain Knowledge

Notice also that in this ontology we used operational relations (marked as dashed arrows) besides the normal compositional ones. This would provide a deeper comprehension of how the concepts interact between them in the domain.

The core of the knowledge base represented in Figure 1, describes that a coordination pattern is a part of a *coordination dependence graph* (CDG) [18] abstracting code which is seen as a composition of statements concerned with coordination aspects, and are used to analyze architectures. As novelty the web of knowledge shows that coordination patterns communicate with each other through ports.

From this description, and knowing that the main objective of CoordL is to define a graph over the composition of statements in the source code of a system, then some kind of graph representation need to be embodied in the language. An obvious reference for representing graphs in a textual way is the DOT language [6], so, CoordL borrows some aspects from that language. The notion of communication ports (in and out) came from the ACME language [7], although the notion of *ports* is very different in these two contexts. To know which ports exist for a pattern, it was adopted the notion of *arguments* from any *general-purpose programming language* (GPL). The description of what are these ports led to the introduction of declarations and initializations in the language. Declarations describe the types of statements represented as nodes in the graph, while the initialization describes the service call which is performed by the node.

From this textual description we defined a syntax by means of a context free grammar shown (partially) in Listing 1.1.

Listing 1.1. Partial grammar for CoordL

```
1  lang → pattern+
2  pattern → ID '(' ports '|' ports ')' '{' decls graph '}'
3  ports → lstID
4  decls → (decl ';')+
5  decl → 'node' lstID '=' nodeRules | 'fork' lstID | 'join' lstID |
6   'ttrigger' lstID | ID instances
7  instances → instance (',' instance)*
8  instance → ID '(' ports '|' ports ')'
9  ...
10 graph → aggregation | connections
11 aggregation → patt_ref ('+' patt_ref)*
12 patt_ref → cnode | '(' aggregationn ')' connection
13 ...
14 cnode → node | ID '.' propTT
15 ...
16 connection → '{' operations '}' '@' '[' ports_alive '|' ports_alive ']'
17 ...
18 operation → cnode link cnode | fork | join | ttrigger
19 ...
20 fork → node sp_link '{' cnode ',' cnode '}'
21 ...
22 link → '−' ID '−' '>'
23 ...
```

Figure 2 presents two examples of patterns written with CoordL. The pattern a) is known as the *Asynchronous Sequential Pattern* which is a pattern used when the system has to invoke a series of services but the order of the answer is not important. The pattern b) is known as the *Joined Asynchronous Sequential Pattern*, which is a transformation of the first pattern to impose order in the responses.

Both of these patterns address different aspects of the syntax, but the main structure of the patterns is the same. Moreover, they address the composition and reuse of patterns.

Regard, for instance, pattern in Figure 2.a). It has a unique identifier (pattern_-1) and declares in and out ports, identifiers p_0 and p_1, p_2 and p_3 respectively. The in ports go on the left side of the '|' (bar) symbol, and the out ports on its right. Then, a space is reserved for node declarations and initializations. There are 5 types of nodes in CoordL: node, fork, join, ttrigger and pattern instance. In Figure 2.a) we use the node and fork types, and in Figure 2.b) we use node, join and pattern instance types. The ttrigger type is similar to fork or join.

Nodes of type *node* require an initialization where it is described a list of rules addressing the corresponding coordination code fragment, the type of interaction or the calling discipline. These rules are composed using the $\&\&$ (and) and/or $||$ (or) logical operators, and the list must, at least, embody one of the following: (i) Statement (st), presents the code fragment of the statement responsible by the coordination request. This statement may be described by a regular expression or may be a complete sentence; (ii) Call Type (ct), defines the type of service requested. The options are not limited, but some of the most used are web services, RMI or Remoting; (iii) Call Method (cm), defines the method in which the request is made. It can be either synchronous or asynchronous and (iv) Call Role (cr), describes the role of the component that is requesting the service. It can be either consumer or producer.

```
1  pattern_1 (p0 | p1, p2, p3){
2    node p0, p3 { st == "*" }
3    node p1, p2 {
4      st == "calling(*)" &&
5      ct == webservice &&
6      cm == sync &&
7      cr == consumer &&
8    };
9    fork f1, f2;
10
11   {f2 -(x,w)-> (p3, p2)}
12   {f1 -(x,y)-> (f2, p1)}
13   {p0 -x-> f1}
14 }
```

(a)

```
1  pattern_2 (p1 | p2){
2    node p1, p2, pa = {st == "*"};
3    pattern_1 patt(i1 | o1, o2, o3);
4    join j1, j2;
5
6    (p1 + patt + p2)
7    {p1 -x-> patt(i1),
8    (patt(o1), patt(o3)) -(x,y)-> j1}
9    {( j1, patt(o2)) -(x,w)-> j2}
10   {j2 -x-> p2}
11 }
```

(b)

Fig. 2. Definition of Two Coordination Patterns with `CoordL`

Pattern instance nodes have the type of an existent pattern. In Figure 2.b), line 3, it is declared an instance of pattern `pattern_1`. Each instance of a pattern must be initialized with unique identifiers referring to all the in and out ports of the pattern typing it.

In `CoordL`, a node is seen as a *pseudo-pattern* (with in and out ports, which may be the node itself). Any operation over these *pseudo-patterns* define a new *pseudo-pattern*. The main operations are the aggregation and the connection. Aggregation[3] is the combination of two or more *pseudo-patterns* by putting them *side-by-side*, this is, not connecting them. The syntax for the aggregation operation is at line 6 of Figure 2.b). Connection is the combination of two nodes by means of an edge with the identification of, at least, a running thread. Examples may be seen in lines 11, 12 and 13, of `pattern_1` and 7, 8, 9 and 10 of `pattern_2`.

These two operations are used to build the graph of the pattern, which comes after all the node declarations. There are two ways of defining the graph: (i) the implicit composition, where there are only used connection operations and (ii) the explicit composition,

[3] Aggregation may be used alone, but will never define a usable pattern.

where aggregation and connection operations are used simultaneously. The graph of the `pattern_1` uses implicit composition, while `pattern_2` uses explicit composition.

The connection operation uses one or more out nodes and one or more in nodes (depending on the type of in and out nodes). When the connection uses these nodes, their implicit in or out ports are closed, meaning that no newer connection can use these nodes as in or out ports again. But, as sometimes it is needed to reuse a node as a in or an out port of a connection, it is needed to re-open them to be used in the sequent connections. This is done with the '@' (alive) operator.

We acknowledge that with all the operators and the associated syntax, the code of the pattern is not easily readable. This way, we defined a visual notation with a suitable "translation" into the textual notation of `CoordL`. In Figure 3 we present the components of the visual notation corresponding to the textual elements that define the graph. In Figure 4 we present how the patterns in Figure 2 look like in this notation.

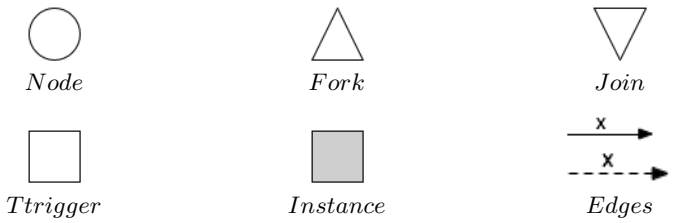


Fig. 3. Components of the Visual Notation for `CoordL`

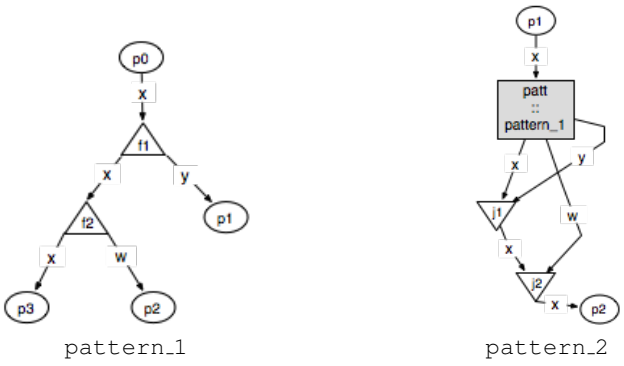


Fig. 4. Visual Representation of Two Coordination Patterns

4 CoordL - The Semantics

The constructs presented in Section 3 have a precise meaning in `CoordL`. In some cases it is possible to draw a mapping between the meaning of a construct and the dependence graph which is extracted from the source code of the system being analyzed. The following paragraphs explain, using informal textual description, the semantics of each construct in the language.

Bar: $|$

This construct separates a list of identifiers into two. The identifiers on the left side list are called in ports and those on the list at the right side are called out ports. It may appear in the *signature* of the pattern, or in the graph of the pattern, when it is needed to keep ports opened for further use.

Aggregation: $pp_1 + pp_2$

This construct sets two *pseudo-patterns* side by side but do not connect them. This is used to re-inforce the existence of the *pseudo-patterns* in the graph, before connect their ports.

Connection: n_1 –x–> n_2

This construct creates a link between two nodes in the graph of the pattern. It means that in the dependence graph $\mathcal{G}$, where the pattern will be applied, there is a path of one or more edges going from n_1 to n_2 through one or more edges in a thread identified by x.

Fork Connection: f –(x, y)–> (n_1, n_2)

This construct creates a link between two nodes in the graph of the pattern, where the start node is a fork. It means that in the dependence graph $\mathcal{G}$, where the pattern will be applied, there are two parallel paths (p_1 and p_2) going from f to n_1 through one or more edges in a thread identified by x, and from f to n_2 in a freshly spawned thread identified by y, respectively. A necessary pre-condition is that in the dependence graph, there is some path p_0 from any node to f in a thread identified by x.

Join Connection: (n_1, n_2) –(x, y)–> j

This construct creates a link between two nodes in the graph of the pattern, where the end node is a join. It means that in the dependence graph $\mathcal{G}$, where the pattern will be applied, there are two parallel paths (p_1 and p_2) going from n_1 to j through one or more edges in a thread identified by x, and from n_2 to j in a thread identified by y, respectively. A necessary pre-condition is that in the dependence graph, there are two paths (p_0 and p'_0) from a fork node to n_1 in a thread identified by x and from the same fork node to n_2 in a thread identified by y, respectively.

Thread Trigger Connection: (n_1, n_2) –(x, y)–> $tt.sync$, (n_1, n_2) –(x, y)–> $tt.fail$

This construct creates a link between two nodes in the graph of the pattern, where the end node is a ttrigget. It means that in the dependence graph $\mathcal{G}$, where the pattern will be applied, there are two parallel paths (p_1 and p_2) going from n_1 to tt through one or more edges in a thread identified by x, and from n_2 to tt in a thread identified by y, respectively. This meaning is replied to express what happens when the threads synchronize (tt.sync), or when the threads synchronization fails (tt.fail). A necessary pre-condition is that in the dependence graph there are two paths (p_0 and p'_0) from a fork node to n_1 in a thread identified by x and from the same fork node to n_2 in a thread identified by y, respectively.

List of Connections: $\{\ connection, \ldots\ \}$
This construct creates a list of independent connections. That is, a connection inside that list do not depend on any node, node property or even on other connections that are used and defined in the list. This independence dues to the fact that there is no order between the connections inside a list of connections. Sequent lists of connections may, but are not obliged to, depend on previous lists.

Along with this construct comes the notion of *fresh nodes*. A fresh node is a control node (like a fork, join or ttrigger) that is firstly used in a connection, and cannot be reused in the same list because of the order dependence. For instance, a fork node must be used as an out port in a connection before being used as a in node.

Alive: @
This construct instructs that a list of identifiers are kept alive as in and out ports. Ports need to be reopened because once a connection uses a node, the implicit port of such node is *killed*. The '@' construct is followed by a list of identifiers divided into two by the bar construct.

5 CoordL - Compiling & Transforming

We used `AnTLR` system [15] to produce a parser for `CoordL`. Taking advantage of the `AnTLR` features we adopted a *separation of concerns* method to generate the full-featured compiler. Figure 5 shows the architecture of the compiler system. The main piece of the compiler system is the syntax module where we specified both the concrete and abstract syntax for `CoordL`, using the context free grammar presented in Listing 1.1. Based on the abstract syntax, `AnTLR` produces an intermediate structure of that grammar known as a tree-grammar.

From the tree-grammar (using attribute grammars methodology) we were able to define new modules that do not care about the concrete syntax. These modules embody the semantics checker, the graph drawer and the unimaginable number of possible transformations applied to that tree-grammar.

The following hierarchical dependence on these modules is observed: the semantics module depends on the syntax module; the graph drawer and the transformation modules depend on the semantics module, so, for transitivity, they also depend on the syntax module. This holds the requirement that some modules may only be used if the syntax and the semantics of the `CoordL` sentence are correct.

We reckon that the separation of concerns on the modules and the dependence between them may be seen as a problem to maintain the compiler. For instance, if something in the abstract syntax of the language changes, these changes must be performed in every dependent module. Nevertheless, this method brings also positive aspects: (i) the number of code lines in each file decreases, easing the comprehension of the module for maintenance; (ii) since each module defines an operation over the coordination patterns code, the compiler may be integrated in a software system providing independent features to manipulate the patterns and (iii) the separation of concerns into modules would ease the maintenance of each feature.

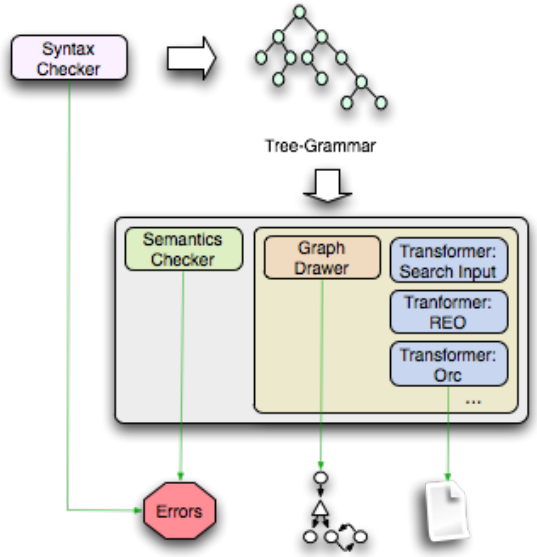


Fig. 5. `CoordL` Compiler Architecture

The transformation modules have, as main objective, to provide perspectives about the coordination patterns, namely, their transformation into Orc [14] or REO [3] specifications. An important module to be considered is the transformation of the pattern code into a suitable input to search for these patterns in the dependence graph of a system code. As for the syntax and the semantic modules, their main output is the syntactic and semantic errors, respectively. The graph drawer module outputs the visual representation of the coordination patterns.

6 Applications and Further Work

The list of possible applications of `CoordL` is not very long. Its precise objective of discovering coordination patterns in a dependence graph reduces its applicability into other areas. Nevertheless, the area of architectural analysis and comprehension allows a profound application of this language.

`CoordInspector` [17] is a tool to extract the coordination layer of a system to represent it in suitable visual ways. In a fast overview, `CoordInspector` processes *common intermediate language* (`CIL`), meaning that systems written in more than 40 `.NET` compliant languages can be processed by that tool. The `CIL` code is then transformed into an `SDG` which is sliced to produce a `CDG`. Then it is used *ad-hoc* graph notations and rules to perform a blind search for non-formalized patterns in the `CDG`. Here is where `CoordL` has its relevance. Due to its systematization and robust formal semantics, the process of discovering patterns in the code would be more reliable than using

the *ad-hoc* rules. The integration of `CoordL` in `CoordInspector` led to the development of an editor to deal with the language. The editor makes heavy use of the `CoordL` compiler system, namely the syntax and semantics modules in order to check whether there are or there are not errors in the patterns specification.

`CoordInspector` is used for integration of complex information systems, resorting to the discovering of coordination patterns. The use of `CoordL` in this task is crucial for a faster and systematized search for such parts of code.

In order to avoid the repetition of writing recurrent patterns, we decided to create a repository of coordination patterns. The repository may be accessed by means of web services from the editor in `CoordInspector`. This repository is not yet finished, but the main objective is to give developers and analysts the possibility of expressing recurrent coordination problems in a `CoordL` pattern and documenting them with valuable information. The existence of the repository of coordination patterns and the fact of being possible the definition of a *calculus* over the language, allows the creation of relations between the patterns, defining an order of patterns.

As further work we may think about how a *calculus* over the language would allow the development of a model checker for analyzing the properties of these patterns.

7 Conclusion

In this paper we introduced a domain-specific language named `CoordL`. This language is used to describe coordination patterns for posterior use in discovering and extracting recurrent coordination code compositions in the tangled source code of a software system.

We explained how the language was designed resorting to (i) the application domain description, by means of an ontology, and (ii) existing programming language and associated knowledge. Then we showed how we took advantage of `AnTLR` to define a full-featured and concern-separated compiler for the language. The adoption of this systematic development of modules for the compiler and the dependencies between them may arise some discussions about the flexibility at maintenance phase. We are aware of such problems, nevertheless we argue that the separation of concerns by modules allow a better use of the compiler when integrated in other tools, and the problems of maintenance are not that numerous because the comprehension of the modules is easier due to having a small number of lines of code, and the issue solved in these lines is known *a priori*.

Finally, we argue for the applicability of `CoordL` along with `CoordInspector`, a tool to aid in architectural analysis and systems reengineering, and the creation of a pattern repository for (i) cataloguing of valuable information about these coordination patterns and (ii) allowing their adoption reuse by developers and analysts.

References

1. Jonathan Aldrich, Craig Chambers, and David Notkin. Archjava: connecting software architecture to implementation. In *ICSE '02: Proceedings of the 24th International Conference*

on Software Engineering, pages 187–197, New York, NY, USA, 2002. ACM.
2. Robert Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon, School of Computer Science, January 1997.
3. Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical. Structures in Comp. Sci.*, 14(3):329–366, June 2004.
4. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, Chichester, UK, 1996.
5. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.
6. Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exper.*, 30(11):1203–1233, 2000.
7. David Garlan, Robert T. Monroe, and David Wile. Acme: Architectural description of component-based systems. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of Component-Based Systems*, pages 47–68. Cambridge University Press, 2000.
8. A. Joseph Goguen. Reusing and interconnecting software components. *Computer*, 19(2):16–28, 1986.
9. George T. Heineman and William T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
10. S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, volume 23, pages 35–46, New York, NY, USA, July 1988. ACM.
11. Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
12. Lih-ren Jen and Yuh-jye Lee. IEEE Recommended Practice for Architectural Description of Software-intensive Systems. *IEEE Architecture*, pages 1471–2000, 2000.
13. David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, and Walter Mann. Specification and analysis of system architecture using rapide. *IEEE Trans. Softw. Eng.*, 21(4):336–355, 1995.
14. Misra, Jayadev, Cook, and William. Computation orchestration: A basis for wide-area computing. *Software and Systems Modeling (SoSyM)*, 6(1):83–110, March 2007.
15. Terence Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. The Pragmatic Bookshelf, Raleigh, 2007.
16. Aoun Raza, Gunther Vogel, and Erhard Plödereder. Bauhaus - a tool suite for program analysis and reverse engineering. In *Reliable Software Technologies - Ada-Europe 2006*, pages 71–82. LNCS (4006), June 2006.
17. Nuno Rodrigues. *Slicing Techniques Applied to Architectural Analysis of Legacy Software*. PhD thesis, Engineering School, University of Minho, October 2008.
18. Nuno F. Rodrigues and Luis S. Barbosa. Slicing for architectural analysis. *Science of Computer Programming*, March 2010.
19. Kamran Sartipi, Lingdong Ye, and Hossein Safyallah. Alborz: An interactive toolkit to extract static and dynamic views of a software system. In *ICPC '06: Proceedings of the 14th IEEE International Conference on Program Comprehension*, pages 256–259, Washington, DC, USA, 2006. IEEE Computer Society.
20. Margaret-Anne Storey. Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal*, 14(3):187–208, September 2006.
21. Robert Tairas, Marjan Mernik, and Jeff Gray. Using ontologies in the domain analysis of domain-specific languages. *Models in Software Engineering*, pages 332–342, 2009.
22. Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. *ACM SIGPLAN Notices*, 35:26–36, 2000.

GammaPolarSlicer

A Contract-based Tool to help on Reuse

Sérgio Areias, Daniela da Cruz, Pedro Rangel Henriques, and Jorge Sousa Pinto

Departamento de Informática e CCTC
Universidade do Minho
Braga, Portugal

Abstract. In software development, it is often desirable to reuse existing software components. This has been recognized since 1968, when Douglas Mcllroy of Bell Laboratories proposed basing the software industry on reuse. Despite the failures in practice, many efforts have been made to make this idea successful.
In this context, we address the problem of reusing annotated components as a rigorous way of assuring the quality of the application under construction. We introduce the concept of *caller-based slicing* as a way to certify that the integration of an annotated component with a contract into a legacy system will preserve the behavior of the former.
To complement the efforts done and the benefits of the slicing techniques, there is also a need to find an efficient way to visualize the annotated components and their slices. To take full profit of visualization, it is crucial to combine the visualization of the control/data flow with the textual representation of source code. To attain this objective, we extend the notion of System Dependence Graph and slicing criterion.

1 Introduction

Reuse is a very simple and natural concept, however in practice it is not so easy. According to the literature, selection of reusable components has proven to be a difficult task [1]. Sometimes this is due to the lack of maturity on supporting tools that should easily find a component in a repository or library [2]. Also, non experienced developers tend to reveal difficulties when describing the desired component in technical terms. Most of the times, this happens because they are not sure of what they want to find [2, 3]. Another barrier is concerned with reasoning about component similarities in order to select the one that best fits in the problem solution; usually this is an hard mental process [1].

Integration of reusable components has also proven to be a difficult task, since the process of understanding and adapting components is difficult, even for experienced developers [1]. Another challenge to component reuse is to certify that the integration of such component in a legacy system is correct. This is, to verify that the way the component is invoked will not lead to an incorrect behavior.

A strong demand for formal methods that help programmers to develop correct programs has been present in software engineering for some time now. The Design by Contract (DbC) approach to software development [4] facilitates modular verification

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 137–148

and certified code reuse. The contract for a component (a procedure) can be regarded as a form of enriched software documentation that fully specifies the behavior of that component. So, a well-defined annotation can give us most of the information needed to integrate a reusable component in a new system, as it contains crucial information about some constraints safely obtaining the correct behavior from the component.

In this context, we say that the annotations can be used to verify the validity of every component's invocation; in that way, we can guarantee that a correct system will still be correct after the integration of that component. This is the motivation for our research: to find a way to help on the safety reuse of components.

This article introduces GamaPolarSlicer, a tool that we are currently developing to identify when an invocation is violating the component annotation, and display, whenever possible, a diagnostic or guidelines to correct it. For such a purpose, the tool implements the **caller-based slicing** algorithm, that takes into account the calls of an annotated component to certify that it is being correctly used.

The remainder of this paper is structured into 5 sections. Section 2 is devoted to basic concepts. In this section the theoretical foundation for GamaPolarSlicer is settle down; the notions of caller-based slicing and annotated system dependence graph are defined. Section 3 gives a general overview of GamaPolarSlicer, introducing its architecture; each block on the diagram will be explained. Sub-section 3.1 complements the architecture discussing the decisions taken to implement the tool and presenting the interface underdevelopment. Section 4, also a central one, illustrates the main idea through a concrete example. As to our knowledge we do not known any tool similar to GamaPolarSlicer, in Section 5 we discuss related work concerned with the use of slicing technique for annotated programs. Then the paper is closed in Section 6.

2 Basic Concepts

We consider that each procedure consists of a body of code, annotated with a precondition and a postcondition that form the procedure specification, or *contract*. The body may additionally be annotated with loop invariants. Occurrences of variables in the precondition and postcondition of a procedure refer to their values in the pre-state and post-state of execution of the procedure respectively.

2.1 Caller-based slicing

In this section, we briefly introduce our slicing algorithm.

Definition 1 (Annotated Slicing Criterion) *An* annotated slicing criterion *of a program $\mathcal{P}$ consists of a triple $C_a = (a, S_i, V_s)$, where $a \in \{\alpha, \delta\}$ is an annotation of $\mathcal{P}_a$ (the annotated callee), S_i correspond to the statement of $\mathcal{P}$ calling $\mathcal{P}_a$ and V_s is a subset of the variables in $\mathcal{P}$ (the caller), that are the actual parameters used in the call and constrained by α or δ.*

Definition 2 (Caller-based slicing) *A* caller-based slice *of a program $\mathcal{P}$ on an annotated slicing criterion $C_a = (\alpha, call_f, V_s)$ is any subprogram $\mathcal{P}'$ that is obtained from $\mathcal{P}$ by deleting zero or more statements in a two-pass algorithm:*

1. *a first step to execute a backward slicing with the traditional slicing criterion $C = (call_f, V_s)$ retrieved from C_a — $call_f$ corresponds to the call statement under consideration, and V_s corresponds to the set of variables present in the invocation $call_f$ and intervening in the precondition formula (α) of f*
2. *a second step to check if the statements preceding the $call_f$ statement will lead to the precondition satisfaction of the callee;*

For the second step in the two-pass algorithm, in order to check which statements are respecting or violating the precondition we are using abstract interpretation, in particular symbolic execution.

According to the original idea of *James King* in [5], symbolic execution can be described as "instead of supplying the normal inputs to a program (e.g. numbers) one supplies symbols representing arbitrary values. The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols."

Using symbolic execution we will be able to propagate the precondition of the function being called through the statements preceding the call statement. In particular, to integrate symbolic execution with our system, we are thinking in use JavaPathFinder [6]. JavaPathFinder is a tool than can perform program execution with symbolic values. Moreover, JavaPathFinder can mix concrete and symbolic execution, or switch between them. JavaPathFinder has been used for finding counterexamples to safety properties and for test input generation.

The main goal of our caller-based slicing algorithm is to ease the use of annotated components by discovering statements that are critical for the satisfaction of the precondition or postcondition (i.e, that do not verify it, or whose value can lead to the non-satisfaction) before or after calling an annotated procedure (a tracing call analysis of annotated procedures). In the work reported here, we just deal with preconditions and statements before the call.

2.2 Annotated System Dependence Graph (SDGa)

In this section we present the definition of Annotated System Dependence Graph, SDG_a for short, that is the internal representation that supports our slicing-based code analysis approach. We start with some preliminary definitions.

Definition 3 (Procedure Dependence Graph) *Given a procedure $\mathcal{P}$, a Procedure Dependence Graph, PDG, is a graph whose vertices are the individual statements and predicates (used in the control statements) that constitute the body of $\mathcal{P}$, and the edges represent control and data dependencies among the vertices.*

In the construction of the PDG, a special node, considered as a predicate, is added to the vertex set: it is called the *entry* node and is decorated with the procedure name.

A control dependence edge goes from a predicate node to a statement node if that predicate affects the execution of the statement. A data dependence edge goes from an assignment statement node to another node if the variable assigned at the source node is used (is referred to) in the target node.

Additionally to the natural vertices defined above, some extra assignment nodes are included in the PDG linked by control edges to the entry node: we include an

assignment node for each formal input parameter, another one for each formal output parameter, and another one for each returned value — these nodes are connected to all the other by data edges as stated above. Moreover, we proceed in a similar way for each call node; in that case we add assignment nodes, linked by control edges to the call node, for each actual input/output parameter (representing the value passing process associated with a procedure call) and also a node to receive the returned values.

Definition 4 (System Dependence Graph) *A System Dependence Graph, SDG, is a collection of Procedure Dependence Graphs, PDGs, (one for the main program, and one for each component procedure) connected together by two kind of edges: control-flow edges that represent the dependence between the caller and the callee (an edge goes from the call statement into the entry node of the called procedure); and data-flow edges that represent parameter passing and return values, connecting actual$_{in,out}$ parameter assignment nodes with formal$_{in,out}$ parameter assignment nodes.*

Definition 5 (Annotated System Dependence Graph) *An Annotated System Dependence Graph, SDG$_a$, is a SDG in which some nodes of its constituent PDGs are annotated nodes.*

Definition 6 (Annotated Node) *Given a PDG for an annotated procedure $\mathcal{P}_a$, an Annotated Node is a pair $< S_i, a >$ where S_i is a statement or predicate (control statement or entry node) in $\mathcal{P}_a$, and a is its annotation: a pre-condition α, a post-condition ω, or an invariant δ.*

The differences between a traditional SDG and a SDG$_a$ are:

- Each procedure dependence graph (PDG) is decorated with a precondition as well as with a postcondition in the entry node;
- The *while* nodes are also decorated with the loop invariant (or true, in case of invariant absence);
- The *call* nodes include the pre- and postcondition of the procedure to be called (or true, in case of absence); these annotations are retrieved from the respective PDG and instantiated as explained below;

We can take advantage from the *call linkage dictionary* present in the SDG$_a$ (inherited from the underlying SDG) — the mapping between the variables present in the call statement (the actual parameters) and the formal parameters of the procedure — to associate the variables used in the calling statement with the formal parameters involved in the annotations.

3 GamaPolarSlicer — Architecture and Implementation

As referred previously, our goal is to ease the process of incorporating an annotated component into an existent system. This integration should be smooth, in the sense of that it should not turn a correct system into an incorrect one.

To assure this, there is the need to verify a set of conditions with respect to the annotated component and its usage:

– Verify its correctness with the respect to its contract (using a traditional *Verification Condition Generator*, already incorporated in GamaSlicer [7], available at `http://gamaepl.di.uminho.pt/gamaslicer`);
– Given a concrete call to the reusable component, verify if the concrete calling context preserves the precondition;
– Given a concrete call and the postcondition of the component, verify if it is properly used in the concrete context after the call;
– Given a reusable component and a set of calling points, specify the component body according to the concrete calling needs.

The chosen architecture to achieve the second condition was based on the classical structure of a language processor. Figure 1 shows the defined GamaPolarSlicer architecture. Notice that the third and fourth conditions will be addressed by future projects.

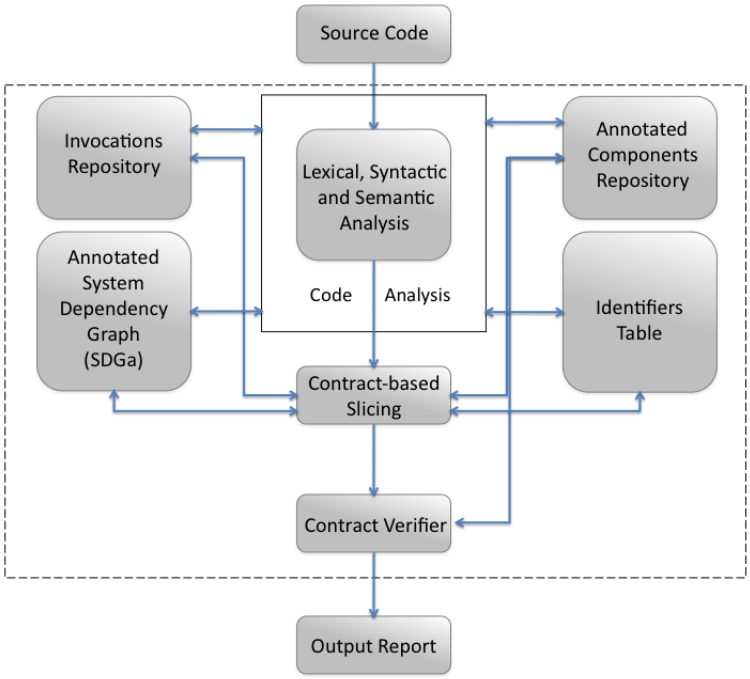


Fig. 1. GammaPolarSlicer Architecture

Source Code is the input to analyze.

Lexical Analysis, Syntactic Analysis, Syntactic Analysis: the Lexical layer converts the input into symbols that will be later used in the identifiers table. The Syntactic layer uses the result of the Lexical layer above and analyzes it to identify the syntactic structure of it. The Semantic layer adds the semantic information to the result from the Syntactic layer. It is in this layer that the identifier table is built.

Invocations Repository is where all invocations found on the input are stored in order to be used later as support to the slicing process.

Annotated Components Repository is where all components with a formal specification (precondition and post condition at least) are stored. It is used in the slicing process only to filter the invocations (from the invocation repository) without any annotation. Has an important role when verifying if the invocation respects component's contract.

Identifiers Table has an important role on this type of programs as always. All symbols and associated semantic found during the analysis phase are stored here. It will be one of the backbones of all structures supporting the auxiliary calculations.

Annotated System Dependence Graph is the intermediate structure chosen to apply the slicing.

Caller-based Slicing uses both invocations repository and annotated components repository to extract the parameters to execute the slicing for each invoked annotated component. The resulting slice is a SDG_a this a subgraph of the original SDG_a.

Contract Verification using the slice that resulted from the layer above, and using the component contract, this layer analyzes every node on the slice and verifies in all of them if there are guarantees that every annotation in the contract is respected.

Output Report presents a view of all violations found during the whole process to the user. In a later stage of this project, exists the possibility of also present suggestions to solve them.

3.1 Implementation

To address all the ideas, approaches and techniques presented in this paper, it was necessary to choose the most suitable technologies and environments to support the development.

To address the *design-by-contract* approach we decide to use the Java Modeling Language (JML) [1]. JML is a formal behavior interface specification language, based on *design-by-contract* paradigm, that allows code annotations in Java programs [8]. JML is quite useful as it allows to describe how the code should behave when running it. Also it allows the specification of the syntactic interface [8]. Preconditions, postconditions and invariants are examples of formal specifications that JML provides.

As the goal of the tool is not to create a development environment but to support one, our first thought was to implement it as an Eclipse [2] plugin. The major reasons that led to this decision were:

[1] http://www.cs.ucf.edu/ leavens/JML/

[2] http://www.eclipse.org/

– the large community and support. Eclipse is one of the most popular frameworks to develop Java applications and thus a perfect tool to test our goal;
– the fact that it includes a great environment to develop new plugins. The Plugin Development Environment (PDE) [3] that allows a faster and intuitive way to develop Eclipse plugins;
– the built-in support for JML, freeing us from checking the validity of such annotations.

However, the parser generated for Java/JML grammar exceeded the limit of bytes allowed to a Java class (65535 bytes). Thus, this limitation led us to abandon the idea of the Eclipse plugin and implement GamaPolarSlicer using Windows Forms and C# (using the .NET framework).

Figure 2 shows the current interface of GamaPolarSlicer.

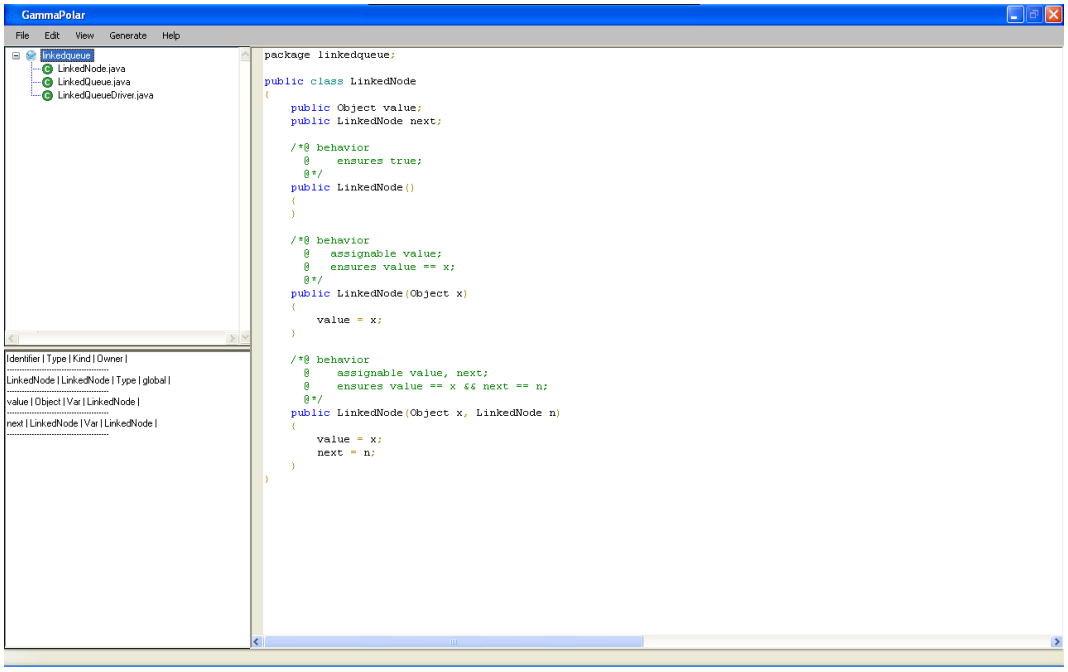


Fig. 2. Interface of GamaPolarSlicer prototype

4 An illustrative example

To illustrate what we intended to achieve, please consider the Example 1 listed below that computes the maximum difference among student ages in a class. This component

[3] http://www.eclipse.org/pde/

reuses other two: the annotated component `Min`, defined in Example 2, that returns the lowest of two positive integers; and Max, defined in Example 3, that returns the greatest positive integer.

Example 1 DiffAge

```
 1: public int DiffAge() {
 2:       int min = System.Int32.MaxValue;
 3:       int max = System.Int32.MinValue;
 4:       int diff;
 5:
 6:       System.out.print(”Number of elements: ”);
 7:       int num = System.in.read();
 8:       int[] a = new int[num];
 9:       for(int i=0; i¡num; i++) {
10:            a[i] = System.in.read();
11:       }
12:
13:       for(int i=0; i¡a.Length; i++) {
14:            max = Max(a[i],max);
15:            min = Min(a[i],min);
16:       }
17:
18:       diff = max - min;
19:       System.out.println(”The gap between max and min age is ” + diff);
20:       return diff;
21: }
```

Example 2 Min

/ ∗ @ requires $x \geq 0$ && $y \geq 0$
@ ensures $(x > y)$? \result $== x$: \result $== y$
@ ∗ /

```
1: public int Min(int x, int y) {
2: int res;
3: res = x − y;
4: return ((res < 0)? x : y);
5: }
```

Let us consider that we want to analyze the `Min` invocation present in the `DiffAge` component.

Our slicing criterion will be: $C_a = (x \geq 0 \&\& y \geq 0, Min, \{a[i], min\})$

In the second step, a backward slicing process is performed taking into account the variables present in V_s. Then, using the obtained slices, the detection of contract

Example 3 Max

$/*$ @ requires $x \geq 0$ && $y \geq 0$
@ ensures $(x > y)?$ \result $== y$: \result $== x$
@ $*/$

```
1: public int Max(int x, int y) {
2: int res;
3: res = x − y;
4: return ((res > 0)? x : y);
5: }
```

violations starts. For that, the precondition is back propagate (using symbolic execution) along the slice to verify if it is preserved after each statement. Observing the slice for the variable `a[i]`, listed in the example 4 below, it can not be guaranteed that all integer elements are greater than zero; so a potential precondition violation is detected.

Example 4 Backward Slicing for a[i]

```
int[] a = new int[num];
for(int i=0; i<num; i++) {
    a[i] = System.in.read();
}
for(int i=0; i<a.Length; i++) {
    max = Max(a[i],max);
    min = Min(a[i],min);
}
```

The third step consists in the notification of all the contract violations detected. In the example above, the user will receive a *warning* alerting to the possible invocation of Min with negative numbers (what does not respect the precondition).

In order to help to visualize which contracts and statements are being violated, we display the SDG_a with such entities colored in red. Figure 3 shows a fragment of the SDG_a for the example above.

5 Related Work

In this section we review the published work on the area of slicing annotated programs, as those contributions actually motivate the present proposal.

In [9], *Comuzzi et al* present a variant of program slicing called *p-slice* or *predicate slice*, using Dijkstra's weakest preconditions (wp) to determine which statements will affect a specific predicate. Slicing rules for assignment, conditional, and repetition statements were developed. They presented also an algorithm to compute the minimum slice.

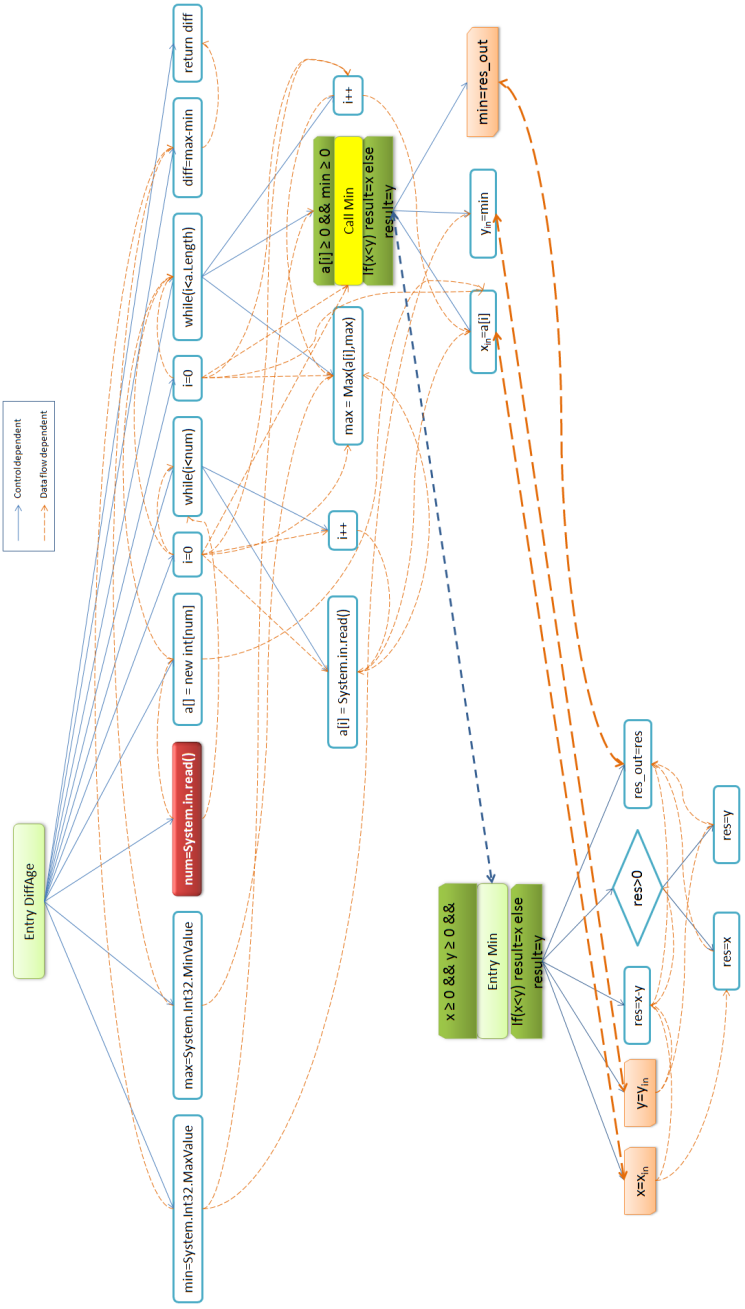


Fig. 3. Example of an Annotated System Dependence Graph

In [10], *Chung et al* present a slicing technique that takes the specification into account. They argue that the information present in the specification helps to produce more precise slices by removing statements that are not relevant to the specification for the slice. Their technique is based on the weakest pre-condition (the same present in *p-slice*) and strongest post-condition — they present algorithms for both slicing strategies, backward and forward.

Comuzzi et al [9], and *Chung et al* [10], provide algorithms for code analysis enabling to identify suspicious commands (commands that do not contribute to the post-condition validity).

In [11], *Harman et al* propose a generalization of conditioned slicing called pre/-post conditioned slicing. The basic idea is to use the pre-condition and the negation of the post-condition in the conditioned slicing, combining both forward and backward conditioning. This type of program slicing is based on the following rule: "Statements are removed if they cannot lead to the satisfaction of the negation of the post condition, when executed in an initial state which satisfies the pre-condition". In case of a program which correctly implements the pre- and post-condition, all statements from the program will be removed. Otherwise, those statements that do not respect the conditions are left, corresponding to statements that potentially break the conditions (are either incorrect or which are innocent but cannot be detected to be so by slicing). The result of this work can be applied as a correctness verification for the annotated procedure.

6 Conclusion

As can be seen in section 4, the motivation for our research is to apply slicing, a well known technique in the area of source code analysis, to create a tool that aids programmers to build safety programs reusing annotated procedures.

The tool under construction, GamaPolarSlicer, was described in Section 3. Its architecture relies upon the traditional compiler structure; on one hand, this enables the automatic generation of the tool core blocks, from the language attribute grammar; on the other hand, it follows an approach in which our research team has a large knowhow (apart from many DSL compilers, we developed a lot of Program Comprehension tools: Alma, Alma2, WebAppViewer, BORS, and SVS). The new and complementary blocks of GamaPolarSlicer implement slice and graph-traversal algorithms that have a sound basis, as described in Section 2; this allows us to be confident in there straight-forward implementation.

GamaPolarSlicer will be included in Gama project (for more details see `http://gamaepl.di.uminho.pt/gama/index.html`). This project aims at mixing specification-based slicing algorithms with program verification algorithms to analyze annotated programs developed under Contract-base Design approach. GamaSlicer is the first tool built under this project for intra-procedural analysis that is available at `http://gamaepl.di.uminho.pt/gamaslicer/`.

Although reuse was not the topic of the paper (just some considerations were drawn in the Introduction), reuse is the main motivation for GamaPolarSlicer development. We are preparing an experiment to assess the validity of our proposal and the usefulness of the tool.

References

1. Maiden, N.A.M., Sutcliffe, A.G.: People-oriented software reuse: the very thought. In: Advances in Software Reuse - Second International Workshop on Software Reusability, IEEE Computer Society Press (1993) 176–185
2. Sherif, K., Vinze, A.: Barriers to adoption of software reuse a qualitative study. Inf. Manage. **41**(2) (2003) 159–175
3. Shiva, S.G., Shala, L.A.: Software reuse: Research and practice. In: ITNG, IEEE Computer Society (2007) 603–609
4. Meyer, B.: Applying "design by contract". Computer **25**(10) (1992) 40–51
5. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7) (1976) 385–394
6. Anand, S., Păsăreanu, C.S., Visser, W.: Jpf-se: a symbolic execution extension to java pathfinder. In: TACAS'07: Proceedings of the 13th international conference on Tools and algorithms for the construction and analysis of systems, Berlin, Heidelberg, Springer-Verlag (2007) 134–138
7. da Cruz, D., Henriques, P.R., Pinto, J.S.: Gamaslicer: an online laboratory for program verification and analysis. In: Proceedings of the 10th Workshop on Language Descriptions Tools and Applications (LDTA'10). (2010)
8. Leavens, G.T., Cheon, Y.: Design by contract with jml (2004)
9. Comuzzi, J.J., Hart, J.M.: Program slicing using weakest preconditions. In: FME '96: Proceedings of the Third International Symposium of Formal Methods Europe on Industrial Benefit and Advances in Formal Methods, London, UK, Springer-Verlag (1996) 557–575
10. Chung, I.S., Lee, W.K., Yoon, G.S., Kwon, Y.R.: Program slicing based on specification. In: SAC '01: Proceedings of the 2001 ACM symposium on Applied computing, New York, NY, USA, ACM (2001) 605–609
11. Harman, M., Hierons, R., Fox, C., Danicic, S., Howroyd, J.: Pre/post conditioned slicing. icsm **00** (2001) 138

Identification and Characterization of Crosscutting Concerns in MATLAB Systems[1]

Miguel P. Monteiro[1], João M. P. Cardoso[2], Simona Posea[3]

[1, 3] CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
[1]mmonteiro@di.fct.unl.pt, [3]simona8810@yahoo.com
[2] Departamento de Engenharia Informática, Faculdade de Engenharia (FEUP), Universidade do Porto, 4200-465 Porto, Portugal, [2]jmpc@acm.org

Abstract. The current state-of-the-art in aspect mining is well advanced for object-oriented programming languages but until now neglected the MATLAB language. This paper contributes to fill that gap by proposing a novel notion of crosscutting concern, tailored for the specific characteristics of MATLAB code bases. We present an exploratory, token-based, approach to aspect mining for MATLAB. An analysis of data obtained from a tool using this approach over 209 publicly available MATLAB files indicate the approach is valid for detecting several kinds of crosscutting concerns in MATLAB systems.

Keywords: MATLAB, aspect mining, code tangling, crosscutting concerns.

1 Introduction

Currently, the state-of-the-art of *aspect mining* [6], i.e., identifying and locating crosscutting concerns (CCCs) as latent aspects in source code, is well advanced for object-oriented (OO) programming languages such as Java. However, most research on aspect mining has until now neglected the popular and widely used procedural language MATLAB. We contribute to address this gap by presenting an exploratory effort to identify and characterize CCCs in MATLAB systems. This paper argues that, due to the specific characteristics and different typical uses of MATLAB, a rethink of the notion of crosscutting concern is warranted, which also requires fresh approaches for their identification in MATLAB code bases.

This paper highlights some differences between symptoms of the presence of CCCs in OO source code and MATLAB code and proposes a simple but effective, token-based, approach for identifying and locating CCCs in MATLAB code. We present an exploratory study of CCCs in a number of real MATLAB applications in the domain of signal and image processing, as well as a short analysis of data collected from that study.

[1] Work for the present publication was partially supported by project AMADEUS under grant (POCTI, PTDC/EIA/70271/2006) from FCT (Portuguese Science Foundation).

The rest of this paper is organized as follows. Section 2 reviews traditional notions of CCC. Section 3 outlines our approach to tackle CCCs in the specific case of MATLAB and on that basis proposes a novel notion of CCC, tailored for MATLAB. Section 4 analyses the suitability of current of aspect mining techniques to be used as an initial, exploratory approach for aspect mining for MATLAB systems. Section 4 proposes an adapted version of one of those techniques for that purpose. Section 5 analyses data collected from MATLAB systems in the domain of signal and image processing. Section 6 outlines opportunities for future work and concludes the paper.

2 Crosscutting concerns and aspect-oriented programming

It has long been accepted that existing programming paradigms suffer from the *tyranny of the dominant decomposition* [12], meaning that each programming paradigm provides a *single* decomposition criterion for a software system. As a consequence, concerns that do not align with the primary decomposition tend to cut across the decomposition units. Such non-aligning concerns are known as *crosscutting concerns* [8]. The usual symptoms of the presence of a CCC in source code are *code scattering* and *code tangling* [8].

Aspect-oriented programming [8] was proposed as an approach to modularize CCCs and thus eliminate the negative symptoms of scattering and tangling. The majority of aspect-oriented languages are backwards-compatible extensions of existing languages, with a marked predominance of OO languages such as Java. Typically, such language extensions add a distinct kind of (often class-like) module – the *aspect* – for the specific purpose of enclosing code related to CCCs.

In OO systems, decomposition units are classes as full-fledged modules. In such systems, code scattering usually takes the form of code fragments scattered across multiple units of modularity, often corresponding to repeated instances of "boiler plate code". Tangling is found in the modules that the CCCs overlap: code pertaining to the primary concern appears intertwined with code pertaining to other concerns. Tangling is particularly harmful to the comprehensibility of all concerns found in the unit, including the primary concern. A Java example is given in section 3.

In less structured programming paradigms such as that supported by (mainly procedural) MATLAB, decomposition units are simpler, less powerful constructs. Since the MATLAB language features are very different from those available in typical OO languages, it is reasonable to expect that symptoms of CCCs and indeed the very idea of CCC requires some adaptation to the different context.

In principle, any approach to identify and locate CCCs is directly dependent of whatever notions are held of what is a CCC. Such notions are traditionally based on the capabilities of the specific aspect-oriented extension of the language concerned. However, that approach assumes an already existing, clearly defined AOP extension to the language. The present situation differs somewhat for two reasons: (1) the distinct characteristics of MATLAB (see section 3), and (2) the fact that in this case, development of the infrastructure to support the language extension is ongoing [4]. The design of our infrastructure enables a wide range of choices in its future development, which can be geared to tackle whatever turns out to make a promising

aspect. Our primary criterion for selection is *impact*: the uses that promise to be useful to most MATLAB users comprise the most desirable targets for future development.

The work presented in this paper is part of an ongoing project whose primary aim is to create an aspect-oriented extension to the MATLAB programming language [5]. The approach taken is focused on the support to separate *aspect modules* that specify functionality that would otherwise cut across, or "pollute", the core parts of the system (MATLAB functions) giving rise to the tangling/pollution symptom.

A detailed description of the infrastructure and underlying approach is not required to understand our proposed notion of CCC. For such an description we refer to our DSAL 2010 paper [4]. It suffices to know the following:

- A *transformational approach* is taken, in which *aspect weaving*, i.e., composition of the aspect modules to the core parts of the application, is carried out by transformation tools according to rules specified in the aspect modules. The outcome of the aspect weaving is transformed, legal MATLAB code.
- The approach is more invasive than traditional aspect-oriented approaches such as AspectJ [7]. Virtually any detail in the base code is potentially the target of some transformation. The transformed code can be considerably different from the code from the base, original, system.
- Potential uses for the transformations supported cover a wide range of concerns, from monitoring, profiling, debugging, to the configuration of variables to support non-standard *shapes*, i.e., numerical representations.
- The infrastructure was designed to support the addition of new, *composition rules* that facilitate a routine development of *case-specific*, *throw-away aspects*.

This approach markedly differs from the usual, compiled approaches to AOP. It also differs from AspectMatlab [1], an extension of MATLAB specifically developed for scientific programming. Our infrastructure targets a significantly wider range of domains. Its transformational approach is motivated by the specific characteristics of MATLAB, particularly its interpreted nature, which blurs the usual distinction between *source* code and *executable* code. MATLAB code plays both roles: as a parallel with Java, the MATLAB code generated by the infrastructure can be regarded as a product of *source-code transformation*, but it is equally reasonable to approach it as the outcome of *bytecode weaving* [7].

3 A notion of crosscutting concerns in MATLAB

In addition to its interpreted nature, the characteristics of MATLAB differ from languages typically extended to aspect-orientation in other respects that also have an influence on the specific details of the symptoms of the presence of CCCs. The procedural nature of MATLAB means that its decomposition units are predominantly individual functions and groups of functions called toolboxes. MATLAB's OO extensions comprise a recent add-on: typical MATLAB applications do not use them. Since our primary criteria for the selection of problems to be tackled is impact, issues pertaining to MATLAB's OO features are not promising candidates. It is worth noting that OO features are also ineffective in modularizing the concerns that are the focus of this paper, often for the same reasons why they also appear in OO systems.

Simple functions and groups of functions comprise a less powerful, more limited mechanism for modularizing concerns than class modules, so it is to be expected that concerns will not appear so well organized within MATLAB systems. Typical uses of MATLAB also differ from those of typical OO languages, so they may give rise to different code symptoms. Therefore, studies of the actual code symptoms in existing MATLAB systems are warranted, namely to characterize the precise ways in which they differ from symptoms in OO code, as well as the typical underlying causes. To our knowledge, no such studies have been carried out until now. The sole prior description of symptoms of the presence of CCCs in MATLAB code is provided by Cardoso et al [5]. Their example comprises the configuration of a benchmark to a specific fixed-point representation. The description provided includes snippets of a clean, version of a function and a "polluted" version using a specific fixed-point representation, programmatically supported. The example suggests fine-grained shape configuration may be a typical CCC in MATLAB.

To illustrate the proposed notion of CCCs in MATLAB and highlight differences between symptoms of CCCs in OO languages and symptoms of CCCs in MATLAB systems, we next describe a simple example of CCC in each platform. In both cases, the secondary concern is the graphical presentation of data, i.e., a *display concern*. However, the specific details for each case are markedly different, reflecting the differences between the Java and MATLAB platforms, as well as their typical uses.

3.1 A crosscutting concern in Java

Fig. 1 shows the classes comprising an example often used to illustrate the effects of CCCs in Java systems, including in the seminal paper on AspectJ [7]. It comprises an abstract declaration of a `FigureElement` type, plus a number of classes – `Point` and `Line` – that concretize it. The classes provide operations for changing the values of the figure elements. Also included is a display functionality to provide a graphical presentation of the figure elements that must be updated upon all changes to them. This is represented by class `Display`, also shown in Fig. 1.

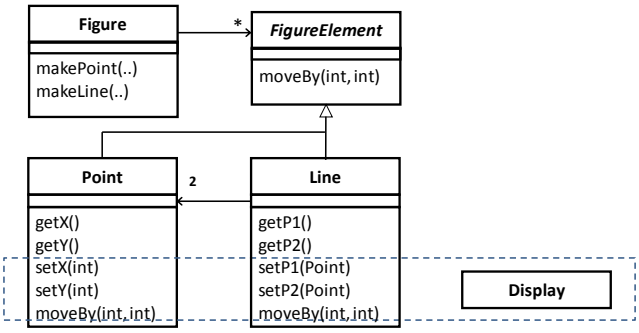


Fig. 1. The Figure example used in [7]

The problem is felt when trying to compose the update logic to the existing decomposition units (here: classes). All operations that have an impact on the

graphical representation of the figures must undergo invasive changes, in the form of the references to `Display`. Fig. 2 shows the effect on source code of class `Point`, with code pertaining to the secondary concern highlighted in grey background. We see code related to the display concern in addition to code related to the module's core concern. Similar symptoms would be found in other `FigureElement` classes (not shown). Note that in real cases, multiple CCCs are often found in single modules.

```
public class Point implements FigureElement {
  private int _x, _y;
  private Display _display;

  public Point(int x, int y) {
     _x = x;
     _y = y;
  }
  public Point(int x, int y, Display display) {
     this(x, y);
     setDisplay(display);
  }
  public void setX(int x) {
     _x = x;
     _display.update(this);
  }
  public void setY(int y) {
     _y = y;
     _display.update(this);
  }
  public void setDisplay(Display display) {
     _display = display;
  }
  public void moveBy(int dx, int dy) {
     _x += dx;
     _y += dy;
     _display.update(this);
  }
}
```

Fig. 2. Illustrative example of a CCC in Java

3.2 A crosscutting concern in MATLAB

Fig. 3 shows a MATLAB function in two versions. The value returned for a given parameter value is the same in both cases. At the left a clean version is shown, whose code relates to its core concern exclusively – compute the result of the exponential function applied to a specific parameter value using the first N terms of the power series expansion. At the right, a tangled version is shown also including a display concern, MATLAB style: preparing a call to function 'plot'. The code pertaining to the secondary concern is highlighted in grey, as in Fig. 2. The extra code is mostly about building the vector data required for building a two-dimensional representation of the function within a given range, which will feed 'plot' at the end. The tangled version defines an additional (and of optional use) parameter to support a choice between creating and not creating the plot representation. The sole motivation for this

parameter is support to one of the primary advantages of modularity: (un)pluggability of the functionality concerned. Note that in real-world examples, use of 'plot' would pollute the original code even more, as programmers usually also include a title and legends in the plotted figure.

This example differs from the Java example in one crucial point: the secondary concern does not merely comprise additional code, intertwined with the original code (i.e., tangling). Rather, additional functionality has a direct impact on the original code that yields *different, modified code*. The computation is no longer performed on simple, scalar values. Instead, vectors are used in place of the original variables to produce the data that will feed 'plot'.

Recent versions of MATLAB incorporate object-oriented features. However, it is not possible to reuse the original version from this example, even through inheritance.

This MATLAB example is of course a trivial one, having been taken from tutorial material used for teaching programming at the institution to which one author is affiliated. However, the course is not controlled by the authors and the example is presented as is. That such a clear example of a CCC can be found in tutorial material serves to highlight how pervasive this kind of symptoms is in MATLAB systems.

```
function z = expo(x,n)
  y = 1;
  for i = 1:n
    y = y + x^i/factorial(i);
  end
  z = y;
```

```
function z = expo(x,n,p)
  P(1) = 1;  Y(1) = 1;
  for i = 1:n
    P(i+1) = P(i)+1;
    Y(i+1) = Y(i) + x^i/factorial(i);
  end
  z = Y(n+1);
  if (p) plot(P,Y) end
```

Fig. 3. Illustrative example of a CCC in MATLAB

3.3 A notion of crosscutting concerns in MATLAB

This paper builds upon the work by Cardoso et al [5] and proposes a notion of a CCC in MATLAB that covers any case in which a given decomposition unit – for brevity, henceforth just "function" – encloses code that can conceptually be traced to more than one concern. The stress on the conceptual (semantic) level is important: a CCC in a function is always a *secondary* concern that is found in addition to the function's *core concern*. In all such cases, the secondary concern can conceivably be extracted from the function, yielding a "clean" version of the function in which only code pertaining to the core concern remains. However, in MATLAB systems it must be assumed that the presence of a secondary concern yields *different* code from the original, which also directly depends on the specifics of the secondary concern.

It is important to note that a significant part of the secondary concerns that fall into the proposed definition can conceivably be implemented by means of some additional language feature – such as can be supported by our infrastructure. Thus, a promising indicator of a potential aspect is to feel the need or desirability of some unsupported additional feature, of a possibly narrow applicability. Though narrowly case-specific,

modularization and (un)pluggability of such features may bring significant benefits to developers.

Discovering some of these concerns can be a subtle task. For instance, Fig. 4 shows a fragment from one of the systems we analyzed. ('drawedgelist'). It is used to assist debugging and requires an assignment to variable 'debug', defined in the MATLAB code of the function with a default value of 0 (i.e., `false`).

We identified the following preliminary list of CCC categories upon manual inspection of a number of real cases, complemented with insights acquired upon analysis of data collected from a tool (presented in the next section):

- *Messages and monitoring*: messages to the user, warnings, errors, graphics visualization, monitoring, etc.;
- *I/O data*: reading data from file, writing data to file, saving an image, loading an image, etc.;
- *Verification of function arguments and return values*: default shapes and values for the arguments that may not be passed in certain function calls;
- *Data type verification and specialization*: check whether a variable is of certain type, configuring the assignment of data types to variables, etc.
- *System*: code that verifies certain system environment properties, to pause execution, etc.
- *Memory allocation/deallocation*: The use of the 'zeros' function is most of times used to allocate a specific array size. This avoids the reallocation for each new item to be stored in an array. Use of the 'clear' instruction that appears in some MATLAB functions is another example.
- *Parallelization*: use of parallel primitives such as 'parfor';
- *Design space exploration*: code to explore different specializations, different algorithms to solve the same problem, to find the number of iterations needed (e.g., to be above a certain precision).
- *Dynamic properties*: constructing inline function objects (*inline*), executing a string containing MATLAB expressions ('eval'), etc.

```
if debug
  for I = 1:Nedge
     mid = fix(length( edgelist{I} ) / 2);
     text( edgelist{I}(mid,2), edgelist{I}(mid,1), sprintf('%d',I) )
...
```

Fig. 4. Segment of MATLAB code for debugging purposes.

In the example presented in Fig. 5 (a fragment of function 'gaborconvolve'), we show a number of aspect candidates whose behavior deals with argument verification ('nargin'), class verification ('isa'), freeing memory ('clear'), and messages to the user ('fprintf'). In this case, code related to debugging can be difficult to automatically expose as an argument variable ('feedback') is used as an option to enable the printing of certain messages in the screen. An advanced aspect mining tool should identify the use of functions to print messages to the screen ('fprintf' in this case) and the code associated with those functions.

```
function EO = gaborconvolve(im, nscale, norient, minWaveLength, mult,
... sigmaOnf, dThetaOnSigma, feedback)
  if nargin == 7
    feedback = 0;
  end
  … % original code removed
  if ~isa(im,'double')
    im = double(im);
  end
  … % original code removed
  clear x; clear y; clear theta; % save a little memory
  … % original code removed
  for o = 1:norient, % For each orientation.
    if feedback
      fprintf('Processing orientation %d \r', o);
    end
  … % original code removed
  end
  if feedback, fprintf('      \r'); end
```

Fig. 5. Illustrative example of CCCs in MATLAB

4 Aspect mining

The task of identifying and locating CCCs in existing code is called *aspect mining* [6]. Typical research on aspect mining covers the development of methods and tools for the automatic or semi-automatic detection and identification of concerns that comprise promising candidates for being extracted to their own aspect modules. Once identification is concluded, a refactoring process can be performed for the extraction of aspects yielding an aspect-oriented version of the original target system [10].

To date, research on aspect mining is largely focused on the OO paradigm, plus the specific case of the C programming language [3]. Java is the most usual OO representative [6]. We have no knowledge of any previous approach on aspect mining for MATLAB systems. Moreover, it turns out that MATLAB code places different, possibly more challenging hurdles for aspect mining tasks.

This section next briefly surveys existing approaches to aspect mining [3], [9], [2], [11]. The intent is not to issue a final judgment on those techniques, even in relation to MATLAB. We aim to select the approach that promises to be the most suitable for an *initial*, *exploratory* study in a topic in which many facets remain unclear.

Two techniques focus on the analysis of method calls, respectively static and dynamic: fan-in analysis [9] and analysis of event traces [2]. Both work at an abstraction level and granularity suitable for object-oriented systems. However, MATLAB does not support the equivalent of full Application Program Interfaces (APIs) in OO systems. Most MATLAB elements available for analysis are minute details of function implementations that are hidden from APIs, such as names of local variables, control structures and names of called functions. Indeed, it is not clear that either technique would be effective for cases like the one described is section 3.

Another technique is the extraction of conceptual knowledge from names in code [11]. Unfortunately, the typical style of naming prevalent in the MATLAB community seems an impediment to that kind of approach. Developer support in MATLAB tools is less rich than that provided for the Java community and lacks features such as code completion and refactoring. Due to such circumstances, programmers tend to use short, cryptic names for parameters and variables, often with a single letter. Function names are longer, but with some exceptions they generally comprise a single word (e.g., 'zeros', 'values') or a cryptic combination of a few shortened words/acronyms (e.g., 'cumtrapz', 'tsearchn'). Thus, high-level, domain concepts are more poorly represented in MATLAB than in Java. For this reason, extraction of conceptual knowledge does not seem promising as an initial approach.

Another option are clone detection techniques [3]. As argued in section 3, the presence of secondary concerns in MATLAB often results in modified code whose details depend on both the original primary logic and the secondary concern in question. This results in higher levels of variability of symptoms in the code, as compared to what is observed in OO systems. One can imagine multiple instances of one same function, each differently "transformed" by a different secondary concern. There is no guarantee that each such instance will bear enough similarities for approaches based on resemblances of code fragments. Thus, clone detection techniques do not generally look suitable for detecting and identifying CCCs in MATLAB code, with one exception.

4.1 Aspect mining tailored to MATLAB

Token-based clone detection techniques are about performing a tokenization of the source code and subsequently use the tokens as a basis for clone detection [3]. Our approach is an adapted version of this kind of clone detection technique, which nevertheless allows us to deal efficiently with all MATLAB examples, regardless of language version. Target systems are assumed to be MATLAB-grammar compliant and no checks for syntactic or grammatical correctness are performed.

The core idea is to count occurrences of function calls in non-comment text lines. The tool decomposes MATLAB source files into sequences of tokens and computes metrics based on those tokens. Distinguishing variable accesses from function calls is not straightforward in MATLAB, as the same syntax is used for both. Thus, the tool builds a list of variable names by extracting them from the function declarations, and by analyzing each individual line to determine if it is a variable assignment. If it is, it registers the *lvalue*, i.e., the name in the left side of the assignment. The list of variable names thus collected is used to filter out the collection of names extracted from the source text. It is assumed that the remaining names are function names. For each separate MATLAB file, a number of metrics is computed, including:

1. Number of times a given function name appears in a given MATLAB file.
2. Number of different function names appearing in a given MATLAB file.

The tool can compute the above metrics for individual `*.m` files, all `*.m` files in a toolbox and all `*.m` files in arbitrary collections of toolboxes. Further functionality was included to ensure that computed data is presented in a form that scales to a fairly large number of systems. Output comprises several tables with this data.

5 Analysis of collected data

Though language processing performed by the tool is not sophisticated, the metrics collected proved very useful. Note that a low number of occurrences cannot count as an useful indicator, as it is perfectly reasonable for a function to call another function several times. However, it became clear that a "high" number of calls is a fairly reliable indicator of tangling. For instance, this metric yields value 8 for the example presented by Cardoso et al [5] (not shown due to space constraints). Determining more precise thresholds is an obvious next step that is left for future work.

We computed the aforementioned metrics for a collection of MATLAB repositories (i.e., applications) in the signal and image processing domains, spanning 17 repositories with 209 MATLAB files and a total of 7,775 lines of code. Fig. 6 shows the number of uses of candidate functions and percentage of the most representative candidates. Function 'size' is the most often used: 194 uses that correspond to 15% of the global uses of the functions identifying aspect candidates. Functions 'error', 'zeros' and 'nargin' are also heavily used (*147×, 11.4%*, *123×, 9.5%* and *89×, 6.9%* respectively). Note that a more advanced analysis would be required to pinpoint the uses of 'zeros' used exclusively to allocate memory, as there are cases in which the function is used to actually initialize a matrix with zero values.

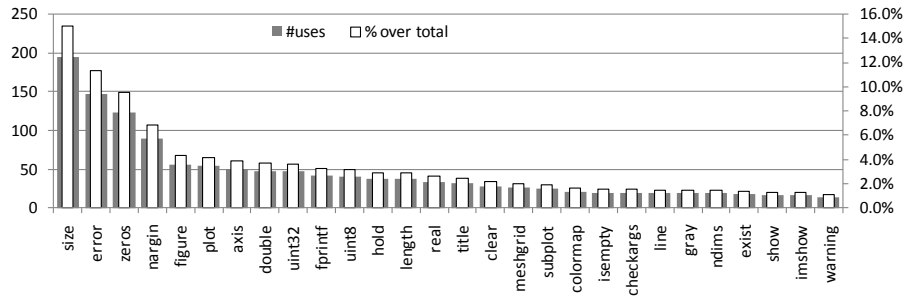


Fig. 6. Aspect candidates present in the analyzed repositories.

The MATLAB files analyzed include the use of 28 functions that we selected as candidates to aspects. Those functions appear 1294 times in the 209 files analyzed, yielding an average of 6.19 uses per `*.m` file. If we measure the "pollution" level as the number of uses per lines of code we have for these examples, around 16.64% of lines use functions that we cataloged as related to aspects. Considering scattering results (i.e., simply considering if a function is used or not in a MATLAB file), there are a total of 646 references which yields about 3.09 different functions referred per file. Fig. 7 shows the frequency of `*.m` files per range of candidate functions. Of 209 files, just 19 do not use candidate functions and 116 files use from 1 to 4 candidate functions. These results indicate a significant proportion of MATLAB code that comprises promising candidates for extraction to future aspectual extensions of MATLAB.

It is worth noting that the MATLAB systems analyzed reveal a low use of shape specialization. This is natural, as most publically available MATLAB systems use high-level models and few of these systems include specialized versions, needed for any case specific use of those systems, e.g., in embedded systems. Most available code is for simulating and problem-solving under the MATLAB environment and is not publicly provided to become part of final system implementations.

Some functions include additional code to make them as generic as possible. When specific specializations are needed, the extra code needed to generalize those functions can be made (un)pluggable, yielding a cleaner version of the core function.

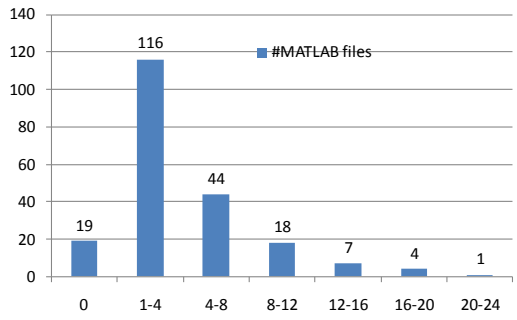


Fig. 7. Frequency of MATLAB files per functions candidate to aspects.

MATLAB models also usually use input/output features during development that will not be used in the final, working system. These include reading from and writing to files, which should be removed and replaced by a suitable input/output aspect.

In some cases, functions are part of a specific behavior and a more elaborate analysis would have to be performed to indentify the code segments that relate to that behavior. For instance, 'nargin' is often used to deal with optional arguments. In the repositories used for this study, a function named 'phasecong' includes 32 lines related to 'nargin' verification and initializations according to the number of arguments passed in the function call. This kind of behavior can conceivably be extracted to a "specialization aspect". That code could then be removed from the MATLAB source and an invocation of the function with fewer arguments could originate a specialization of that function. Such specialization includes the code that should be executed for a certain number of arguments. Another case is when the *class* (i.e., MATLAB type) of a variable is verified and according to the *class* a specific behavior is invoked.

6 Conclusion and future work

This paper described our first contribution to aspect-mining in MATLAB systems, proposing a novel notion of crosscutting concern for MATLAB and evaluating whether instances of those concerns are found in a significant number of MATLAB files. Experimental results using a token-based aspect-mining tool and targeting

publicly available real systems enabled the identification of a number of promising candidates for modularization through extraction to aspect modules.

There are many directions through which this work can be developed in future, contributing to refine and mature the notion of crosscutting concern proposed in this paper. One front is to explore the (semi-)automatic identification of crosscutting concerns ("candidate aspects") on the basis of broader code segments, namely through pattern matching. However, much analysis remains to be performed on data computed through this simple, token-based approach. The "number of calls per individual function" metric proves an useful indicator of tangling, but this hypothesis requires further analysis and assessment. In addition, there are several other hypotheses that can be assessed through this approach, namely: (1) the extent to which "number of functions that call this function" can be a reliable indicator of scattering; (2) whether certain groups of tokens tend to be used together; (3) whether certain tokens tend to appear in connection to specific functionalities; (4) whether specific groups of tokens can be traced to specific application domains.

In a different front, the previously aspect mining techniques surveyed in section 4 should be also explored, to assess in practice the extent to which they can advantageously replace or complement the approach proposed in this paper.

References

1. Aslam T., Doherty J., Dubrau A., Hendren L. AspectMatlab: An Aspect-Oriented Scientific Programming Language. AOSD 2010, Saint Malo, France, March 2010.
2. Breu S., Krinke J. Aspect Mining Using Event Traces. 19th IEEE international conference on Automated software engineering (ASE 2004). Linz, Austria, September 2004.
3. Bruntink M., Deursen A. van, Engelen R. van, Tourwé T. An Evaluation of Clone Detection Techniques for Identifying Cross-Cutting Concerns. ICSM'04, Chicago, Illinois, USA, September 2004.
4. Cardoso, J., Diniz, P., Monteiro, M., Fernandes, J., Saraiva, J. A Domain-Specific Aspect Language for Transforming MATLAB Programs. DSAL 2010, Rennes, March 2010.
5. Cardoso, J., Fernandes, J., Monteiro, M. 2006. Adding Aspect-Oriented Features to MATLAB. SPLAT!2006, Bonn, Germany, March 2006.
6. Kellens A., Mens K., Tonella P. A Survey of Automated Code-Level Aspect Mining Techniques. Transactions on Aspect Oriented Software Development, vol. 4 (LNCS 4640), pp. 145-164. Springer, 2007.
7. Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., Griswold W., An Overview of AspectJ. ECOOP 2001, Budapest, Hungary, June 2001.
8. Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C., Loingtier J., Irwin J., Aspect-Oriented Programming. ECOOP'97, Jyväskylä, Finland, June 1997.
9. Marin M. Deursen A., Moonen L., Identifying Aspects using Fan-In Analysis. Working Conference on Reverse Engineering (WCRE2004), Delft, The Netherlands, Nov 2004.
10. Monteiro M.P., Fernandes J.M. Towards a Catalogue of Refactorings and Code Smells for AspectJ. LNCS TAOSD I, Springer vol. 3880, 2006.
11. Shepherd D., Fry Z., Hill E., Pollock L., Vijay-Shanker K. Using Natural Language Program Analysis to Locate and Understand Action-Oriented Concerns. AOSD 2007, Vancouver Canada, March 2007.
12. Tarr P., Ossher H., Harrison W., Sutton Jr., S.M., N Degrees of Separation: Multi-Dimensional Separation of Concerns. ICSE'99, Los Angeles, USA, May 1999.

Producing EAM code from the WAM

Paulo André[1] and Salvador Abreu[1]

Departamento de Informática,
Universidade de Évora and CENTRIA FCT/UNL, Portugal
{prla,spa}@di.uevora.pt

Abstract. Logic programming provides a very high-level view of programming, which comes at the cost of some execution inefficiency. Improving performance of logic programs is thus one of the holy grails of Prolog system implementations and a wide range of approaches have historically been taken towards this goal. Designing computational models that both exploit the available parallelism in a given application and that try hard to reduce the explored search space has been an ongoing line of research for many years. These goals in particular have motivated the design of several computational models, one of which is the Extended Andorra Model (EAM). In this paper, we present a preliminary specification and implementation of the EAM with Implicit Control, the `WAM2EAM`, which supplies regular WAM instructions with an EAM-centered interpretation.

1 Introduction

Logic programming provides an abstract and high-level view of programming in which programs are expressed as a collection of facts and rules that define a model of the problem at hand and against which questions may be asked. The most well-known example of this paradigm of programming is Prolog, which has been sucessfully used in applications of many different areas. One line of work that has been followed to address performance issues is parallel execution: parallelism allows logic programs to transparently exploit multi-processor environments while extensions like co-routining, constraints and tabling go a long way towards reducing the problem's inherent search space. Some or all of these together act as the foundation on which to build more advanced techniques towards obtaining maximum performance.

From the experience gained in implementing the Basic Andorra Model, D.H.D. Warren made a more radical proposal, the Extended Andorra Model, or EAM [11], where the conditions in which independent computations might be carried out are eagerly sought. In this article, we present a concrete implementation of the Extended Andorra Model, the `WAM2EAM`, which differs from other approaches taken in the past because we are compiling straight WAM code into C,[1] adopting an EAM computational model, resorting to GCC extensions.

This paper is structured as follows: Section 2 presents a short survey on the road leading up to our current implementation as far as the EAM is concerned, from the Andorra Principle to the BEAM. Section 3 describes the EAM in more detail and lays

[1] We are targetting C with GCC extensions, such as label values and indirect jumps.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 161–172

down the theoretical groundwork of the WAM2EAM and delves more deeply into its practical implementation from WAM code compilation to the data structures and execution control of the EAM-based generated C code. We conclude with section 4.

2 State of the Art and Related Work

A significant body of research on Logic Programming has been directed towards improving the performance of Prolog. One important line of research towards this goal is the exploitation of the different forms of implicit parallelism, present in Prolog programs. Several approaches have been devised over the years but we shall focus on the systems which allow for the transparent parallel goal execution, in particular the "Andorra" family of languages which includes Andorra-I, AKL and the BEAM.

2.1 The Andorra Principle

David H. D. Warren proposed the **Basic Andorra Model** (BAM),[2] geared towards the execution of logic programs, in which a goal is called *determinate* if it has at most one candidate clause. In this model, deterministic goals should be executed first, thereby reducing the nondeterminate "guesswork" to the minimum possible. Only then, once no deterministic goal remains to be executed, should a non-deterministic goal be selected for execution.

A system incorporating the Andorra Principle reduces the search space of logic programs by having deterministic goals execute first and only once, rather than have them re-executed several times in different points of the search space. This behavior is also known as "sidetracking." Also, as a desirable consequence, deterministic goals may generate constraints (bindings) which may further reduce the number of alternatives in other (non-deterministic) goals, possibly even making them deterministic.

Another interesting advantage is how all deterministic goals can execute in parallel, as long as they do not run into binding conflicts. Parallelism in the BAM comes in two flavours:

- *AND-Parallelism* - deterministic goals run in parallel
- *OR-Parallelism* - the exploration of different alternatives to a goal is done in parallel

The BAM may also alter the semantics of programs, in that the order of the solutions for a given goal may be different from that resulting from sequential Prolog execution. This may cause otherwise nonterminating programs to reach a solution.

There are, however, a few issues inherent to this sort of computational model:

- Finding which goals are deterministic can sometimes be difficult as predicates with more than one clause may actually have a single matching clause for a given query.
- Concurrency may break Prolog semantics, for instance by executing a pruning directive (e.g. cut) too early.

[2] Not to be confused with van Roy's Berkeley Abstract Machine, used in the Aquarius Prolog system.

The best-known implementation of the Basic Andorra Model is Andorra-I [3,2]. It exploits OR-parallelism and determinate dependent AND-parallelism while fully supporting Prolog, however, despite good results, the system is limited by the fact that co-routining and AND-parallelism can only be exploited between determinate goals.

Shortly after, Warren went further and proposed the **Extended Andorra Model** (EAM) which improved upon the ideas of the BAM, namely by trying to explore independent AND-parallelism. This lead to a two major approaches:

- **AKL**: The Andorra Kernel Language (AKL) [4,5] was designed by Haridi and Janson and was the course followed at SICS. It concentrated on the idea that a new language was needed, based on the advantages of the EAM, which would subsume both Prolog and committed-choice languages. AKL distinguished itself by featuring an *explicit* control scheme, as programs were written using guarded clauses, where the guard was separated from the body with a sequential conjunction, cut or commit operator.
- **EAM with Implicit Control**: In contrast to AKL, David H. D. Warren and other researchers at Bristol worked towards an implementation of the EAM with implicit control [11]. Its main goal was to take advantage of the Andorra Principle while alleviating the burden on the programmer.

2.2 The BEAM

The Boxed EAM (BEAM) is an implementation of the EAM design with implicit control, developed at University of Porto, Portugal [6,7,8,9]. The beam's initial goal was to prove the feasibility of Warren's design for the EAM, and as a first step it concentrated on the original rewriting rules of the EAM, so formally it was defined through rewrite rules that manipulate AND-OR trees as well as simplification and optimization rules used to simplify the tree and discard boxes. It also made use of a general control strategy, which is used to decide when and how to apply each rule.

The main operations of the BEAM are:

- **Reduction** expands a goal G into an OR-box.
- **Promotion** promotes constraints from an inner AND-box to an outer AND-box.
- **Propagation** propagates constraints from an outer AND-box to the inner boxes.
- **Splitting** distributes a conjunction across a disjunction.

Adding to these are a few simplification and optimization rules, all of which are described in [8].

Apart from AND- and OR-boxes, there's also another kind of box contemplated in the BEAM which is the choice-box. These are special OR-boxes created when the clauses defining a procedure include a pruning operator, generically designated by `%`. The original EAM supports two pruning operators, cut and commit.

The EAM tries to keep the control implicit as much as possible, contrary to AKL for instance. Therefore, in the BEAM, the control decisions are based exclusively on information implicitly extracted from the program. Moreover, one of the main goals of the EAM is to perform the least possible number of reductions to obtain the solutions to a goal. BEAM's control strategy is geared towards this goal.

The BEAM also does not attempt to do all the work by itself, instead relying on the output of an existing Prolog compiler, in this case YAP Prolog. The BEAM was built as an extension to YAP. It differs from the work reported herein in that the BEAM is meant to be an interpreter, whereas `WAM2EAM` takes WAM code and compiles it to C.

Non-termination A central problem found by the developers of the BEAM was a consequence of EAM's execution scheme: as long as they do not bind any (external) variables, the EAM allows the early parallel execution of nondeterminate goals. In the worst case, this may lead to non-termination for certain recursive predicates. The proposed solution was based on both eager non-determinate promotion and tabling which, on the one hand guarantees that the computation ends in programs that have finite solutions and on the other hand, with tabling, allows for the reuse of solutions to goals.

3 The Extended Andorra Model and `WAM2EAM`

The Extended Andorra Model (EAM) is the foundation for the work we carried out with `WAM2EAM`. The idea is to perform as much work as possible in parallel, exploiting all the avaliable forms of parallelism:

- *Or-parallelism*, related to exploring the various alternatives of any given goal.
- *Indendent AND-parallelism*, within a conjunction of goals that do not share any variables.
- *Dependent AND-parallelism*, between goals that *do* share variables.

The main extension of the EAM over the BAM is that non-deterministic goals are allowed to execute in parallel so long as they do not bind any external variables.

Our purpose is to provide a concrete implementation of the EAM with implicit control. It departs from existing work because it compiles regular WAM code into C, using an EAM runtime specification. Therefore, the biggest challenge and arguably the most interesting aspect of this work, is going from one paradigm (Prolog compiled onto the WAM) to a different one (EAM) with a single tool.

Based on a configuration AND-OR tree at all times, the way to evolve this configuration is by using one of several *rewrite* rules on it and an execution control scheme to manage the application of these rules. We do not present the rewrite rules used by `WAM2EAM`, as these are closely related to those presented in [11]. An configuration is made up of nodes, or boxes: AND-boxes and OR-boxes, like in the BEAM and other AND-OR tree-based models. What constitutes these boxes and how they are implemented in `WAM2EAM`is explained more thoroughly in Section. **??**.

The major challenge in `WAM2EAM` certainly is to go from a WAM program and reinterpret it from an EAM point of view. To accomplish that, we take the GNU Prolog's textual WAM output and proceed from there. The idea is to generate C code for an EAM runtime. This entails doing things quite differently from previous work such as WAMCC [1] or B-Prolog [10]. `WAM2EAM` is comprised of two major modules:

1. the compiler, comprising the parser and the C code generator,

2. the runtime, a collection of data structures, logic and execution control that implements the EAM execution model.

The remainder of this section discusses design and implementation of the compiler and runtime.

3.1 Parsing WAM instructions

We used GNU Prolog because its compilation passes are fairly simple and it is easy to materialize the WAM representation of Prolog programs. The following is a snippet of code which is the GNU Prolog WAM representation of a `p(X) :- q(X), r(X).` predicate.

```
predicate(p/1,5,static,private,user,[
    allocate(1),
    get_variable(y(0),0),
    put_value(y(0),0),
    call(q/1),
    put_value(y(0),0),
    deallocate,
    execute(r/1)]).
```

We built a parser for this representation in Bison, which constructs an abstract parse tree of the WAM program.

3.2 C Code Generation

An interesting aspect of `WAM2EAM` is how we take a sequence of instructions intended for the regular WAM and directly re-interpret them in an EAM context, yielding appropriate patterns of target code. Be that as it may, a lot of the WAM instruction set translates as-is to the EAM representation. Simpler instructions, such as `put_value` for instance, are supposed to do exactly the same thing in the WAM and in the EAM and the same goes for indexing instructions like `switch_*`. In a few cases, such as `proceed`, `WAM2EAM` simply disregards the instruction as not being useful in the EAM setting.

At closer inspection of the WAM instruction set, the major difference in paradigm impacting the C code generation concerns the instructions dealing with non-determinism. Whereas the WAM deals with choice points, creating and destroying them as needed, the EAM, by doing away with the WAM's stack-based representation and using an AND-OR tree based configuration instead, deals with OR-boxes when it comes to setting up and exploring alternatives.

Once every detail of the original program has a C representation – an abstract parse tree – the idea is to walk through it and emit a bit of C for each predicate and for every WAM instruction inside it. For each internalized predicate, a block of C code is generated, setting up a new AND-box which contains a suitable number of allocated local variables,[3] binding those variables to its parent OR-box corresponding predicate arguments and defining each of those variables' *home* as the very AND-box that is being created. The output code is generated by this code in the compiler:

[3] The exact number is determined by inspection of the WAM code in the body.

```
emit(8, "a = new_and_box (o, %d, ab_id++);\n", max_var_idx+1);
for (i = 0; i < n; i++)
  emit(8, "bind (a->locals[%d], o->args[%d]);\n", i, i);
for (i = 0; i < max_var_idx+1; i++)
  emit(8, "ASREF(a->locals[%d])->home = a;\n", i);
```

`max_var_idx` reflects the maximum number of variables used in this predicate, accounting for possible temporaries in all of its clauses, potentially a single one if deterministic. Looking now at the C code for a clause with two local variables, it might look something like this:

```
a = new_and_box (o, 2);
bind (a->locals[0], o->args[0]);
bind (a->locals[1], o->args[1]);
```

This allocates a new AND-box with two local variables, as a child of the current OR-box (whose address is kept in `o`) and both of those variables are then immediately bound to whatever are the first two parent OR-box arguments. This creates variable chains across the AND-OR tree, reflecting the same concept found in Prolog clauses where a newer variable might refer to an older one.

A second pass through the WAM instructions for the clause is needed to generate code for each actual WAM instruction by traversing the list built by the parser.

```
while (instrs) {
  print_instr (instrs->head, (*a)->name, n, max_var_idx+1, FALSE)
  instrs = instrs->tail;
}
```

`print_instr` then goes through a large `switch` instruction that finds the appropriate bit of C code to emit for each WAM instruction, having the EAM execution scheme in mind. WAM instructions, which by now we regard as EAM instructions in their own right, are roughly divided in three major groups:

Choice point manipulation These are the `try*`, `retry*` and `trust*` instructions. We no longer think in terms of choice point frames, instead looking at managing non-determinism by way of OR-boxes. A predicate with only one clause consists of an OR-box with a single alternative (and thus a single descendant AND-box) whereas a non-deterministic predicate (ie. having more than one clause) is translated as an OR-box with as many children AND-boxes as there are possible clauses. A more in-depth description of how OR-boxes actually deal with alternatives will be given after we introduce the major data structures used throughout `WAM2EAM`. In practice, an instruction like `try_me_else (L)` (or `retry_me_else (L)`, for that matter) for predicate `q(1)` simply defines the next alternative in the current OR-box, generating the following bit of C code:

```
o->alt = &&P_q_1_C4;
```

Execution Control The `call` and `execute` instructions are responsible for predicate calling, in effect jumping to the appropriate place in the code where to start executing the called predicate. They also need to setup a return address where to get back to when this predicate finishes execution. This is accomplished by emiting a C label and configuring the current AND-box continuation to that label, using GCC's label address extension. With this, once the called predicate is done, it will `proceed` to whatever AND-continuation is available in its AND-box, in effect returning here and resuming execution. The difference between `call` and `execute` is precisely what to do after the called predicate is done with. Whereas in the former case, it simply continues executing whatever is left in the current predicate, the latter means this was the last goal in the current clause and it should look for a continuation above, in the Prolog execution chain. Here's how the `call` instruction is translated to C: For example, the pattern of code generated for calling the goal `q(X)` in our example is:

```
  /* call(q/1) */
  q_enqueue(a->and_conts,&&R1); // setup AND-continuation
  o = new_or_box(a,1);          // create new OR-box
  o->args[0] = a->locals[0];    // preload A registers on the new OR-box
  goto P_q_1;                   // jump to the predicate's code

R1:                             // return label
  /* further code.. */
```

Variable manipulation and unification This type of instructions is also handled quite differently within the EAM. Simple instructions such as `put_value` or `get_variable` are basically the same, but unification needs to be looked at more carefully, as trying to bind variables which are not local to the current AND-box leads to suspension of execution and triggers a search for work, elsewhere in the code. AND-box suspension and the `WAM2EAM` execution scheme will be looked upon in a bit more detail shortly.

3.3 Generated code structure

One important constraint on generated code layout is that we must be able to jump back and forth between different predicates, in order to implement predicate calling and returning. Also, we need to jump to random places in the code when attempting to resume a suspension. As it is illegal to use C's `goto` between different functions,[4] generating one C function per predicate is not an option.

One possible alternative then is to implement the entire program as a single function and delimiting predicates using unique labels. This way, jumping from one point in the code to another remains within the bounds of the one function and correct indentation when emitting the code will hopefully not make it a burden to look at. We also must be careful when jumping to a point of code from out of nowhere, since the correct environment must be replaced, namely the current AND- and OR-boxes. Other than that, all it takes for jumping around the code is the address to jump to and making good use of GCC's labels as values extension.

[4] We may not reenter an existing C stack frame.

```
int program ()
{
  /* ... */
  P_p_1: {
      a = new_and_box(o,1);
      /* ... */
      o = new_or_box(a,1);
      goto P_q_1;

  /* ... */
  P_q_1: {
     a = new_and_box(o,1);
     /* ... */
}
```

3.4 Runtime Data Structures

The runtime half of WAM2EAM is itself broken into two major steps and these are where we significantly depart from the WAM way of doing things and completely focus on EAM. First, executing the C code previously generated by the compiler will incrementally build the *configuration*, an AND-OR tree that gets constructed, modified and pruned as execution of the code proceeds. The way for this to happen is by applying in turn the different AND-OR tree rewrite rules.

The most important data structure in WAM2EAM is the AND-OR tree, or *configuration*. An AND-OR tree is so called because it is composed of AND nodes, corresponding to Prolog clauses and OR nodes, consisting of Prolog goals. It is important to note that no two nodes, or *boxes*, of the same type are directly connected in an AND-OR tree. Moreover, the root is always an OR-box.

AND-boxes They represent clauses, so there is one AND-box in the configuration for every clause in the Prolog source code. So, for instance, a non-deterministic predicate having four different clauses, would consist of four AND-boxes, one for each clause. AND-boxes are a lengthy structure in WAM2EAM in that they play a critical role. They are home to the clause's local variables, they need to keep track of their continuations (e.g. where to find the code for the next goal in the clause once the current goal is done with) and they also may or may not be suspended at any point in time. Finally, promotion also impacts AND-boxes directly, so they also have mechanisms to deal adequately with that. And, of course, they spawn (and in turn descend from) OR-boxes corresponding to the reduction of their body goals.

OR-boxes These represent goals and are created everytime a new goal is executed. Their primary concern is dealing with non-determinism by managing goal alternatives, namely holding an address for the next alternative for the current goal at all times. They also carry the goal's arguments when the goal gets called in order to pass them initially to each clause's AND-box as initial values. OR-boxes thus spawn an AND-box for each clause they invocate.

3.5 Suspensions

As we have seen before, caution must be taken when an attempt to bind a variable is made. Only in case the variable is local to the current AND-box will binding be allowed to occur. Otherwise, the AND-box is said to be suspended on the offending variable and execution proceeds elsewhere, namely to the next alternative in the current OR-box. Execution can only return to this AND-box when certain conditions are met, namely when the variable becomes local to the current AND-box or it gets bound from elsewhere. In the latter case, when the suspension is resumed, the attempted binding that triggered the suspension in the first place is retried and it either checks OK or it fails against the prevailing (earlier) binding.

In order to correctly deal with these situations, we need to wrap instructions wherein a suspension might occur with some code that actually checks for "offending" binding attempts, namely trying to bind a non-local variable. We do this by having every unification instruction check whether the dereferenced variable is already bound and if not, whether it is local or external to the current AND-box. The result of this verification is then returned as a meanigful code to a wrapping `CHECK()` macro, which then acts accordingly. Faced with a unification attempt, the outcome can then be any one of:

BIND_SUSP the variable is not bound yet and it is not local to the current AND-box either. The current AND-box suspends on this variable.

BIND_OK the variable is not bound and it is local, so the binding succeeds.

CHECK_OK the variable is bound and its value is the same as the one being attempted in the binding, so execution may proceed.

CHECK_FAIL the variable is bound and its value differs with the one being tried. The configuration branch rooted in the current AND-box fails and is pruned off the tree.

Because of suspensions, for every non-trivial program it is easy to see that we quickly arrive at what we call a *stuck configuration*, an AND-OR tree where all leaf AND-boxes are suspended. As we don't stop execution anytime a box suspends, it is only when no more code is left to execute that we have a problem. At this time we try to apply one of the rewriting rules, in particular giving priority to determinate rules such as determinate promotion. By promoting an inner AND-box into an outer AND-box, the variables inside it are also promoted which means they become closer to the AND-box where they will actually be local, eventually allowing for bindings to happen or suspensions to resume.

3.6 Deterministic Promotion

As explained in the previous section, actions (or rules) that *contract* the configuration are desirable. On the other hand, expanding goals also expands the configuration, as AND-boxes give way to OR-boxes which in turn give way to more AND-boxes and so forth. Deterministic promotion, being the only rule that eliminates boxes, is highly sought after. This rule is only applicable to OR-boxes with a single alternative.

Implementation-wise, promoting an AND-box context (variables, suspensions and continuations) into another requires maintaining their environments coherent. In other words, if the resulting AND-box contains the union of both sets of locals variables

from the two AND-boxes involved in the suspension, then what was the first variable in the inner (promoted) AND-box is probably no longer the first variable in the outer (resulting) AND-box after promotion. This lends itself to all kinds of mayhem when code still refers `a->locals[0]` (WAM register `X(0)`) when the actual variable is now at `a->locals[1]`.

To cope with this problem, we opted to introduce the concept of AND-box groupings. Each AND node in the configuration is actually a group of one or more complete AND-boxes, forward-connected among themselves by a pointer which indicates the next box in the group. Moreover, every box in the group is also linked to the first - the *head*. This situation is illustrated in figure 1.

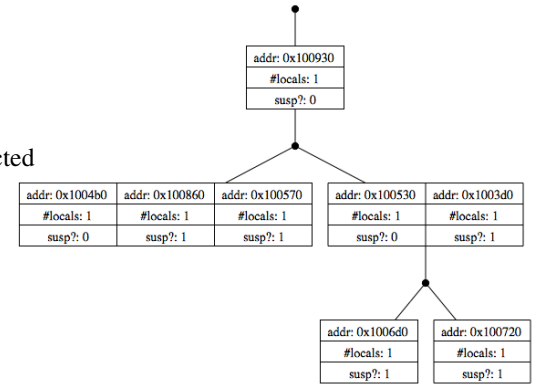


Fig. 1. On the left: an AND-box grouping made of 3 different AND-boxes.

This way, each box environment remains pristine, as originally constructed, and it is safe to resume from a suspension point as far as accessing local variables is concerned. It is important to note that a variable is local to the current AND-box if, after dereferencing, its home AND-box is in the same group, i.e. has the same head.

3.7 OR-split and non-deterministic promotion

Desirable as deterministic promotion might be, its occurence is heavily constrained as we have shown in the previous section. The OR-box must have a single alternative and for predicates with multiple clauses that's frequently not the case. It is quite common for a configuration to get stuck with no chance for deterministic promotions to occur. When it comes to this, there is no other choice than to perform what we call an *OR-split* which forces a situation where a determinate promotion may happen.

Simply put, we elect an OR-box with more than one alternative to act as the root of a subtree to be *cloned*. In the original subtree, only one alternative remains, while in the cloned subtree, *every other* alternative is present. This way, all alternatives remain in the overall configuration, ensuring correctness of the program, yet an opportunity for deterministic promotion now exists. Note that if the selected OR-box contains only two alternatives, we arrive at the special case where the OR-split induces two different deterministic promotion possibilities: one in the original box and another in the cloned box.

The choice of OR-box to split may be guided by heuristics, yet at this early stage we are simply going with the leftmost OR-box suitable for splitting. Also, from the chosen box's alternatives, we are picking the leftmost one to remain in the original branch and all others to be moved to the cloned subtree. Actual cloning is only needed for the parent

AND-box and any siblings of the chosen OR-box. OR-split is the least desirable rule, because with cloning entire branches of the tree, it quickly becomes expensive.

3.8 The scheduler

The need to decide which rule to apply led to the implementation of a scheduler. This scheduler is called the first time after all alternatives and continuations are exhausted and no answers were produced. In other words, when the tree is stuck we ask the scheduler for guidance.

The implementation of the scheduler is part of the runtime code and is implemented as a C macro. It basically follows a hierarchy of possible events and acts accordingly for each outcome. First of all, in the event that a variable that had suspensions got bound, it tries to resume from any suspension pending on that variable. If none are found, it looks for an alternative in the current OR-box. If found, it continues execution from there, otherwise it tests the tree to see if it is stuck. If it is, it tries to apply deterministic promotion in order to try to move on or, if that fails, it resorts to applying non-deterministic promotion, by way of an OR-split. Putting this as the last choice makes sense, because it is also the most expensive operation.

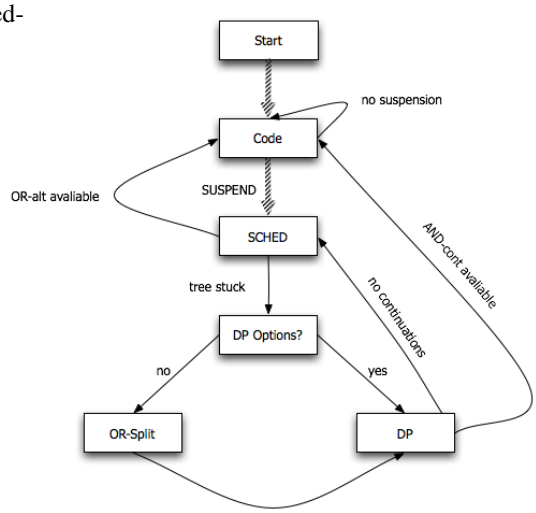


Fig. 2. The scheduler's flow diagram.

It is interesting to note the reason why the scheduler is implemented as a macro instead of a function, despite being a little involved and lengthy, it is because it may involve jumping to any point in the code, be it a suspension point, a continuation or an OR-alternative. Again, we are faced with the problem of not being able to jump between different C functions, so implementing this as a macro solves the problem. The control flow for the scheduler is depicted in figure 2.

4 Concluding Remarks & Future Work

We are convinced that our goal of generating a program following EAM semantics from a classical WAM one has been met, even if with some restrictions for the time being. Performance is not yet an issue but will become one as we develop further aspects of this implementation. It is interesting to see that it is feasible to have an EAM execution model without the Prolog compiler being aware of the fact.

Further work is to focus on the introduction of pruning operators – in the case of *cut* this is straightforward to recognize from the WAM code but for *commit* special measures will have to be taken as it is not inherently accounted for by the Prolog-to-WAM compiler of GNU Prolog.

One of the driving motivations for generating AND-OR trees and having them manipulated as per the EAM was to bridge this computational model to one with tabling, as found in XSB or YAP Prolog. This goal remains in our agenda, as does a parallel version which will be the ultimate test for the claim that AND-OR tree rewriting is a good approach for implicit parallel execution.

Acknowledgments

The authors would like to thank Ricardo Rocha for fruitful discussions on the implementation of `WAM2EAM`. The FCT (Portuguese Government Agency) is acknowledged for supporting this work under the project STAMPA (PTDC/EIA/67738/2006).

References

1. Philippe Codognet and Daniel Diaz. `wamcc`: Compiling Prolog to C. In *12th International Conference on Logic Programming*. The MIT Press, 1995. 3
2. Vítor Santos Costa, David H. D. Warren, and Rong Yang. Andorra-i: A parallel prolog system that transparently exploits both and- and or-parallelism. In *PPOPP*, pages 83–93, 1991. 2.1
3. Vítor Santos Costa, David H. D. Warren, and Rong Yang. The andorra-i engine: A parallel implementation of the basic andorra model. In *ICLP*, pages 825–839, 1991. 2.1
4. Torkel Franzén, Seif Haridi, and Sverker Janson. An overview of the andorra kernel language. In Lars-Henrik Eriksson, Lars Hallnäs, and Peter Schroeder-Heister, editors, *ELP*, volume 596 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 1991. 2.1
5. Sverker Janson and Seif Haridi. Programming paradigms of the andorra kernel language. In *ISLP*, pages 167–183, 1991. 2.1
6. Ricardo Lopes and Vítor Santos Costa. The beam: A first eam implementation. In Maria Chiara Meo and Manuel Vilares Ferro, editors, *APPIA-GULP-PRODE*, pages 425–440, 1999. 2.2
7. Ricardo Lopes, Vítor Santos Costa, and Fernando M. A. Silva. A novel implementation of the extended andorra model. In I. V. Ramakrishnan, editor, *PADL*, volume 1990 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2001. 2.2
8. Ricardo Lopes, Vítor Santos Costa, and Fernando M. A. Silva. On the beam implementation. In Fernando Moura-Pires and Salvador Abreu, editors, *EPIA*, volume 2902 of *Lecture Notes in Computer Science*, pages 131–135. Springer, 2003. 2.2
9. Ricardo Lopes and Vítor Santos Costa. The BEAM: Towards a first EAM Implementation. In *Proceedings of the Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages*, 1997. 2.2
10. Paul Tarau. The BinProlog Experience: Implementing a High-Performance Continuation Passing Prolog Engine. Technical report, BinNet Corp., 1998. 3
11. David H. D. Warren. The Extended Andorra Model with Implicit Control. ICLP90 Preconference Workshop, June 1990. 1, 2.1, 3

Solving Difficult LR Parsing Conflicts by Postponing Them

L. Garcia-Forte and C. Rodriguez-Leon

Departamento de EIO y Computación,
Universidad de La Laguna, Tenerife, Spain
casiano@ull.es,
WWW home page: http://nereida.deioc.ull.es

Abstract. Though yacc-like LR parser generators provide ways to solve shift-reduce conflicts using token precedences, no mechanisms are provided for the resolution of reduce-reduce conflicts. To solve this kind of conflicts the language designer has to modify the grammar. All the solutions for dealing with these difficult conflicts branch at each alternative, leading to the exploration of the whole search tree. These strategies differ in the way the tree is explored: GLR, Backtracking LR, Backtracking LR with priorities, etc. This paper explores an entirely different path: to extend the yacc conflict resolution sublanguage with new constructs allowing the programmers to explicit the way the conflict must be solved. These extensions supply ways to resolve any kind of conflicts, including those that can not be solved using static precedences. The method makes also feasible the parsing of grammars whose ambiguity must be solved in terms of the semantic context. Besides, it brings to LR-parsing a common LL-parsing feature: the advantage of providing full control over the specific trees the user wants to build.

1 Introduction

Yacc-like LR parser generators [1] provide ways to solve shift-reduce mechanisms based on token precedence. No mechanisms are provided for the resolution of reduce-reduce conflicts or difficult shift-reduce conflicts. To solve such kind of conflicts the language designer has to modify the grammar. Quoting Merrill [2]:

> *Yacc lacks support for resolving ambiguities in the language for which it is attempting to generate a parser. It does a simple-minded approach to resolving shift/reduce and reduce/reduce conflicts, but this is not of sufficient power to solve the really thorny problems encountered in a genuinely ambiguous language*

Some context-dependency ambiguities can be solved through the use of lexical tie-ins: a flag which is set by the semantic actions, whose purpose is to alter the way tokens are parsed. But it is not always possible or easy to resort to this kind of tricks to fix some context dependent ambiguity. A more general solution is to extend LR parsers with the capacity to branch at any multivalued entry of the

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 173–184

LR action table. For example, Bison [7], via the `%glr-parser directive` and Elkhound [5] provide implementations of the Generalized LR (GLR) algorithm [4]. In the GLR algorithm, when a conflicting transition is encountered, the parsing stack is forked into as many parallel parsing stacks as conflicting actions. The next input token is read and used to determine the next transitions for each of the top states. If some top state does not transit for the input token it means that path is invalid and that branch can be discarded. Though GLR has been successfully applied to the parsing of ambiguous languages, the handling of languages that are both context-dependent and ambiguous is more difficult. The Bison manual [7] points out the following caveats when using GLR:

> *... there are at least two potential problems to beware. First, always analyze the conflicts reported by Bison to make sure that GLR splitting is only done where it is intended. A GLR parser splitting inadvertently may cause problems less obvious than an LALR parser statically choosing the wrong alternative in a conflict. Second, consider interactions with the lexer with great care. Since a split parser consumes tokens without performing any actions during the split, the lexer cannot obtain information via parser actions. Some cases of lexer interactions can be eliminated by using GLR to shift the complications from the lexer to the parser. You must check the remaining cases for correctness.*

The strategy presented here extends yacc conflict resolution mechanisms with new ones, supplying ways to resolve conflicts that can not be solved using static precedences. The algorithm for the generation of the LR tables remains unchanged, but the programmer can modify the parsing tables during run time.

The technique involves labelling the points in conflict in the grammar specification and providing additional code to resolve the conflict when it arises. Crucially, this does not requires rewriting or transforming the grammar, trying to resolve the conflict in advance, backtracking or branching into concurrent speculative parsers. Instead, the resolution is postponed until the conflict actually arises during parsing, whereupon user code inspects the state of the underlying parse engine to decide the appropriate solution. There are two main benefits: Since the full power of the native universal hosting language is at disposal, any grammar ambiguity can be tackled. We can also expect - since the conflict handler is written by the programmer - a more efficient solution which reduces the required amount of backtracking or branching.

This technique can be combined to complement both GLR and backtracking LR algorithms [6] to give the programmer a finer control of the branching process. It puts the user - as it occurs in top down parsing - in control of the parsing strategy when the grammar is ambiguous, making it easier to deal with efficiency and context dependency issues. One disadvantage is that it requires a comprehensive knowledge of LR parsing. It is conceived to be used when none of the available techniques - static precedences, grammar modification, backtracking LR or Generalized LR - produces satisfactory solutions. We have implemented these techniques in `Parse::Eyapp` [9], a yacc-like LALR parser generator for

Perl [10, 11]. The Perl language is, quoting Paul Hudak's article [12] a "*domain specific language for text manipulation*".

This paper is divided in six sections. The next section introduces the Postponed Conflict Resolution (PPCR) strategy. The following three sections illustrate the way the technique is used. The first presents an ambiguous grammar where the disambiguating rule is made in terms of the previous context. The next shows the technique on a difficult grammar that has been previously used in the literature [7] to illustrate the advantages of the GLR engine: the declaration of enumerated and subrange types in Pascal [13]. The last example deals with a grammar that can not be parsed by any LL(k) nor LR(k), whatever the value of `k`, nor for packrat parsing algorithms [14]. The last section summarizes the advantages and disadvantages of our proposal.

2 The *Postponed Conflict Resolution* Strategy

The *Postponed Conflict Resolution* is a strategy (PPCR strategy) to apply whenever there is a shift-reduce or reduce-reduce conflict which is unsolvable using static precedences. It delays the decision, whether to shift or reduce and by which production to reduce, to parsing time. Let us assume the `eyapp` compiler announces the presence of a reduce-reduce conflict. The steps followed to solve a reduce-reduce conflict using the PPCR strategy are:

1. Identify the conflict: What LR(0)-items/productions and tokens are involved?.
   Tools must support that stage, as for example via the `.output` file generated by `eyapp`. Suppose we identify that the participants are the two LR(0)-items $A \to \alpha_{\uparrow}$ and $B \to \beta_{\uparrow}$ when the lookahead token is `@`.
2. The software must allow the use of symbolic labels to refer by name to the productions involved in the conflict. Let us assume that production $A \to \alpha$ has label `:rA` and production $B \to \beta$ has label `:rB`. A difference with `yacc` is that in `Parse::Eyapp` productions can have *names* and *labels*. In `Eyapp` names and labels can be explicitly given using the directive `%name`, using a syntax similar to this one:

   $$\texttt{\%name :rA } A \to \alpha$$
   $$\texttt{\%name :rB } B \to \beta$$

3. Give a symbolic name to the conflict. In this case we choose `isAorB` as name of the conflict.
4. Inside the *body* section of the grammar, mark the points of conflict using the new reserved word `%PREC` followed by the conflict name:

   $$\texttt{\%name :rA } A \to \alpha \quad \texttt{\%PREC IsAorB}$$
   $$\texttt{\%name :rA } B \to \beta \quad \texttt{\%PREC IsAorB}$$

5. Introduce a `%conflict` directive inside the *head* section of the translation scheme to specify the way the conflict will be solved. The directive is followed by some code - known as the *conflict handler* - whose mission is to modify the parsing tables. This code will be executed each time the associated conflict state is reached. This is the usual layout of the conflict handler:

```
%conflict  IsAorB {
  if (is_A) { $self->YYSetReduce('@', ':rA' ); }
       else { $self->YYSetReduce('@', ':rB' ); }
}
```

   Inside a conflict code handler the Perl default variable `$_` refers to the input and `$self` refers to the parser object.
   Variables in Perl - like `$self` - have prefixes like `$` (scalars), `@` (lists), `%` (hashes or dictionaries), `&` (subroutines), etc. specifying the type of the variable. These prefixes are called *sigils*. The sigil `$` indicates a *scalar* variable, i.e. a variable that stores a single value: a number, a string or a reference. In this case `$self` is a reference to the parser object. The arrow syntax `$object->method()` is used to call a method: it is the equivalent of the dot operator `object.method()` used in most OOP languages. Thus the call

```
$self->YYSetReduce('@', ':rA' )
```

   is a call to the `YYSetReduce` method of the object `$self`.
   The method `YYSetReduce` provided by `Parse::Eyapp` receives a token, like `'@'`, and a production label, like `:rA`. The call

```
$self->YYSetReduce('@', ':rA' )
```

   sets the parsing action for the state associated with the conflict `IsAorB` to reduce by the production `:rA` when the current lookahead is `@`.
   The call to `is_A` represents the context-dependent dynamic knowledge that allows us to take the right decision. It is usually a call to a nested parser for A but it can also be any other contextual information we have to determine which one is the right production.

The procedure is similar for shift-reduce conflicts. Let us assume we have identified a shift-reduce conflict between LR-(0) items $A \to \alpha_{\uparrow}$ and $B \to \beta \uparrow \gamma$ for some token `'@'`. Only steps 4 and 5 change slightly:

4'. Again, we must give a symbolic name to $A \to \alpha$ and mark with the new `%PREC` directive the places where the conflict occurs:

$$\begin{array}{rl} \texttt{\%name :rA} & A \to \alpha \ \ \texttt{\%PREC IsAorB} \\ & B \to \beta \ \ \texttt{\%PREC IsAorB} \ \ \gamma \end{array}$$

5'. Now the conflict handler calls the `YYSetShift` method to set the `shift` action:

```
%conflict  IsAorB {
      if (is_A) { $self->YYSetReduce('@', ':rA' ); }
      else { $self->YYSetShift('@'); }
   }
```

3 A Simple Example

The following example[1] accepts lists of two kind of commands: *arithmetic expressions* like `4-2-1` or one of two *associativity commands*: `left` or `right`. When a `right` command is issued, the semantic of the `'-'` operator is changed to be right associative. When a `left` command is issued the semantic for `'-'` returns to its classic left associative interpretation. Here follows an example of input. Between shell-like comments appears the expected output:

```
$ cat input_for_dynamicgrammar.txt
2-1-1 # left:  0 = (2-1)-1
RIGHT
2-1-1 # right: 2 = 2-(1-1)
LEFT
3-1-1 # left:  1 = (3-1)-1
RIGHT
3-1-1 # right: 3 = 3-(1-1)
```

We use a variable `$reduce` (initially set to `1`) to decide the way in which the ambiguity `NUM-NUM-NUM` is solved. If `false` we will set the `NUM-(NUM-NUM)` interpretation. The variable `$reduce` is modified each time the input program emits a `LEFT` or `RIGHT` command.

Following the steps outlined above, and after looking at the `.output` file, we see that the items involved in the announced shift-reduce conflict are

$$expr \rightarrow expr_{\uparrow} - expr$$
$$expr \rightarrow expr - expr_{\uparrow}$$

and the lookahead token is `'-'`. We next mark the points in conflict in the grammar using the `%PREC` directive (see Figure 1)

[1] For the full examples used in this paper, see the directory `examples/debuggingtut/` in the `Parse::Eyapp` distribution [9]

```
%%
p:
      /* empty */     {}
    | p c             {}
;

c:
      $expr { print "$expr\n" }
    | RIGHT { $reduce = 0}
    | LEFT  { $reduce = 1}
;
```

```
expr:
      '(' $expr ')'  { $expr }
    | %name :M
      expr.left         %PREC lOr
        '-' expr.right  %PREC lOr
          { $left -$right }

    | NUM
;
```

Fig. 1. An Example of Context Dependent Ambiguity Resolution

The *dollar* and *dot* notation used in some right hand sides (rhs) like in `expr.left '-' expr.right` and `$expr` is used to associate variable names with the attributes of the symbols.

The conflict handler `lOr` defined in the header section is:

```
%conflict lOr {
  if ($reduce) {$self->YYSetReduce('-', ':M')}
  else         {$self->YYSetShift('-')}
}
```

If `$reduce` is `true` we set the parsing action to *reduce* by the production labelled `:M`, otherwise we choose the *shift action*.

Observe how PPCR allow us to dynamically change at will the meaning of the same statement. That is certainly harder to do using alternative techniques, either problem specific, like *lexical Tie-Ins* [7], or more general, like GLR [4].

4 Nested Parsing of Unexpended Input and Context

This section illustrates the technique through a problem that arises in the declaration of enumerated and subrange types in the programming language Pascal. The problem is taken from the Bison manual, (see section '*Using GLR on Unambiguous Grammars*') where it is used as a paradigmatic example of when to switch to the GLR engine [7]. Here are some cases:

```
type subrange = lo .. hi;
type enum = (a, b, c);
```

The original language standard allows only numeric literals and constant identifiers for the subrange bounds (`lo` and `hi`), but Extended Pascal (ISO/IEC

10206) [13] and many other Pascal implementations allow arbitrary expressions there. This gives rise to declarations like the following:

```
type subrange = (a) .. b;
```

```
type enum = (a);
```

The corresponding declarations look identical until the '..' token. With normal LALR(1) one-token lookahead it is not possible to decide between the two forms when the identifier 'a' is parsed. It is, however, desirable for a parser to decide this, since in the latter case 'a' must become a new identifier to represent the enumeration value, while in the former case 'a' must be evaluated with its current meaning, which may be a constant or even a function call. The Bison manual considers and discards several potential solutions to the problem to conclude that the best approach is to declare the parser to use the GLR algorithm. To aggravate the conflict we have added the C *comma* operator inside `expr`[2], making room for the generation of declarations like:

```
type subrange = (a, b, c) .. (d, e);
```

```
type enum = (a, b, c);
```

which makes the parsing even more difficult.

Here is our modification of the vastly simplified subgrammar of Pascal type declarations found in [7].

```
%token ID  = /([A-Za-z]\w*)/
%token NUM = /(\d+)/

%left   ','
%left   '-' '+'
%left   '*' '/'

%%

type_decl : 'TYPE' ID '=' type ';'
;

type :
      '(' id_list ')'
    | expr '..' expr
;
```

```
id_list :
      ID
    | id_list ',' ID
;

expr :
      '(' expr ')'
    |  expr '+' expr
    |  expr '-' expr
    |  expr '*' expr
    |  expr '/' expr
    |  expr ',' expr /* new */
    |  ID
    |  NUM
;
```

[2] Perhaps the language designer wants to extend Pascal with lexicographic ranges

When used as a normal LALR(1) grammar, eyapp correctly complains about two reduce/reduce conflicts:

```
$ eyapp -v pascalenumeratedvsrange.eyp
2 reduce/reduce conflicts
```

The generated `.output` file tell us that both conflicts occur in state 11. It also give us the contents of state 11:

```
State 11:
    id_list -> ID . (Rule 4)
    expr -> ID .    (Rule 12)

    ')' [reduce using rule 12 (expr)]
    ')' reduce using rule 4 (id_list)
    '*' reduce using rule 12 (expr)
    '+' reduce using rule 12 (expr)
    ',' [reduce using rule 12 (expr)]
    ',' reduce using rule 4 (id_list)
    '-' reduce using rule 12 (expr)
    '/' reduce using rule 12 (expr)
```

From the inspection of state 11 we can conclude that the two reduce-reduce conflicts occur between productions `id_list -> ID` and `expr -> ID` in the presence of tokens '`)`' and '`,`'. To solve the conflict we label the two involved productions and set the `%PREC` directives:

```
id_list :
      ...
      %name ID:ENUM
      ID                      %PREC rangeORenum
      ...
expr : '(' expr ')'
      ...
    | %name ID:RANGE
      ID                      %PREC rangeORenum
      ...
```

When the conflict point is reached the conflict handler below calls the method `YYLookBothWays(`a, b`)`.

```
%conflict rangeORenum {
  my $s = $self->YYLookBothWays('TYPE', ';');
  if ($s =~ /^TYPE ID = \( ID ( , ID )* \) ;$/x)
       { $self->YYSetReduce([',', ')'], 'ID:ENUM' ); }
  else { $self->YYSetReduce([',', ')'], 'ID:RANGE' ); }
}
```

The substring from the *right sentential form*[3] between `'TYPE'` and `';'` is stored in `$s`. If the string of tokens `$s` in Σ^* conforms to the syntax of an enumerated type

```
/^TYPE ID = \( ID ( , ID )* \) ;$/
```

we set the parsing action to reduce by the production $id\_list \rightarrow ID$ for the two conflictive tokens `','` and `')'`. Otherwise the input represents a range type.

In most cases, as occurs in this example, the nested parsing step required to decide which action must be taken can be accomplished through a simple regular pattern. Nested parsing is extraordinarily eased by the fact that Perl 5.10 standard regular patterns permit the description of context free languages. Even more, modules like `Regexp::Grammar` [15] bring Perl 6 [11] regular patterns to Perl 5, extending Perl 5 regular patterns beyond the capabilities of Packrat parsing [14].

The call `YYLookBothWays`(a, b) returns the string that is the concatenation of the transition tokens in the *stack* after token a followed by the tokens in the *unexpended input* before token b. It does not alter the current parsing position.

To be more precise, suppose that at the time of the call the pair

$$(s_0 X_1 s_1 X_2 s_2 \cdots X_m s_m, a_i a_{i+1} \cdots a_n \$) \tag{1}$$

is the *configuration* of the LR parser. Here $X_j \in \Sigma \cup V$ is a token or a syntactic variable, $a_k \in \Sigma$ are tokens and s_i are the states of the LR automata. Remember that the equation $\delta(s_k, X_{k+1}) = s_{k+1}$ for each k is hold by any configuration, δ being the transition function. This configuration corresponds to the right sentential form

$$X_1 X_2 \cdots X_m, a_i a_{i+1} \cdots a_n \$ \tag{2}$$

which must be in the rightmost derivation from the grammar start symbol being built. The call `YYLookBothWays(`a, b`)` returns

$$X_j \cdots X_m a_i a_{i+1} \cdots a_s \$ \tag{3}$$

where j is the shallowest index in the stack such that $X_j = a$ and s is the nearest index in the unexpended input such that $a_s = b$.

[3] A string $\alpha \in (\Sigma \cup V)^*$ is said a *right sentential form* for a grammar $G = (\Sigma, V, P, S)$ if, and only if, exists a rightmost derivation from the start symbol S to α

5 Conflicts Requiring Unlimited Look-ahead

The following unambiguous grammar can not be parsed by any LL(k) nor LR(k), whatever the value of `k`, nor for packrat parsing algorithms [14].

```
%%
S: x S x | x  ;
%%
```

Though it is straighforward to find equivalent LL(1) and LR(1) grammars (the language is even regular: `/x(xx)*/`), both GLR [4] and Backtrack LR parsers [2] for this grammar will suffer of a potentially exponential complexity in the input size. The unlimited number of look-aheads required to decide if the current `x` is in the middle of the sentence, leads to an increase in the number of branches to explore. The challenge is to make the parser work *without changing* the grammar. Figure 2 shows a solution using PPCR:

```
%conflict isInTheMiddle {
  my $nxs = $self->YYSymbolStack(0,-1, 'x'); # number of visited 'x's
  my $nxr = (unexpendedInput() =~ tr/x//);   # number of remaining 'x's

  if ($nxs == $nxr+1) { $self->YYSetReduce('x', ':MIDx' ) }
  else { $self->YYSetShift('x') }
}

%%
S:
     x  %PREC isInTheMiddle S x
  |  %name :MIDx
     x  %PREC isInTheMiddle ;
```

Fig. 2. Parsing a Non LL(k) nor LR(k) nor packrat grammar

A call `$self->YYSymbolStack(`$a, b, [filter]$`)` returns the list of symbols associated with the parser stack states between a and b. A negative value of a or b refers to the position from the end of the list. Thus, a call like `$self->YYSymbolStack(0,-1)` returns the whole list of symbols in the parsing stack. The optional $filter$ argument can be a string, a closure[4] or a regular pattern.

[4] A closure is a first-class function with free variables that are bound in the lexical environment

When it is a pattern the sublist for which the pattern matches is selected. If it is a closure, it returns the sublist of symbols for which the evaluation of the code is true. If it is a string, as in `$self->YYSymbolStack(0,-1, 'x')`, the sublist of symbols equal to the string is selected. Since the assignment `$nxs = $self->YYSymbolStack(0,-1, 'x')` is evaluated in a scalar context, the length of the resulting sublist is stored in the variable `$nxs`. The copy of the unexpended input - returned by the call to `unexpendedInput()` - is then scanned for `'x'`s and its number is stored in `$nxr`. When `$nxs` equals `$nxr +1` it is time to reduce by $S \to x$. It may seem that this solution cannot be generalized when `'x'` is an arbitrary grammar. Remember however that Perl 5.10 regular patterns can parse any context free grammar [15].

We can now compile the former program `nopackratSolved.eyp` to generate a script `nopackratSolved.pl` containing the parser. When executed with input `xxx` it outputs a description of the abstract syntax tree:

```
$ eyapp -TC -o nopackratSolved.pl nopackratSolved.eyp
$ ./nopackratSolved.pl -t -i -c 'xxx'
S(TERMINAL[x],S(TERMINAL[x]),TERMINAL[x])
```

Option `-T` instructs `eyapp` to automatically insert semantic actions to produce a data structure representing the abstract syntax tree. Option `-C` tells the compiler to produce an executable (by default it produces a class containing the parser). Eyapp provides default lexical analyzer, error handler and main subroutines for the generated program. The default `main` subroutine admits several command line options, like: `-t` (print the AST), `-i` (print the semantic values of the tokens) and `-c arg` (take the input from `arg`).

6 Conclusions

The strategy presented in this paper extends the classic yacc precedence mechanisms with new dynamic conflict resolution mechanisms. These new mechanisms provide ways to resolve conflicts that can not be solved using static precedences. They also provides finer control over the conflict resolution process than existing alternatives, like GLR and backtracking LR. There are no limitations to PPCR parsing, since the conflict handler is implemented in a universal language and it then can resort to any kind of nested parsing algorithm. The conflict resolution mechanisms presented here can be introduced in any LR parsing tools, since they are independent of the implementation language and the language used for the expression of the semantic actions. One disadvantage of PPCR is that it requires some knowledge of LR parsing. Though the solution may be more efficient, it certainly involves more programmer work than branching methods like GLR or backtracking LR.

Acknowledgments

This work has been supported by the EC (FEDER) and the Spanish Ministry of Science and Innovation inside the 'Plan Nacional de I+D+i' with the contract number TIN2008-06491-C04-02. It has also been supported by the Canary Government project number PI2007/015.

References

1. Steven C. Johnson. Yacc: Yet another compiler compiler. In *UNIX Programmer's Manual*, volume 2, pages 353–387. Holt, Reinhart, and Winston, 1979. AT&T Bell Laboratories Technical Report July 31, 1978.
2. Gary H. Merrill. Parsing Non-LK(k) Grammars with Yacc. Software, Practice and Experience 23(8): 829-850 (1993).
3. Kernighan & Ritchie. *The C Programming Language.* Prentice Hall.
4. Tomita, M. (1990). The Generalized LR Parser/Compiler - Version 8.4. In *Proceedings of International Conference on Computational Linguistics (COLING'90)*, pages 59–63, Helsinki, Finland.
5. Mcpeak, Scott. September 2004. Elkhound: A Fast, Practical GLR Parser Generator. `http://scottmcpeak.com/elkhound/`
6. Adrian D. Thurston and James R. Cordy. A Backtracking LR Algorithm for Parsing Ambiguous Context-Dependent Languages. 2006 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON 2006), pp. 39-53, Toronto, October 2006.
7. Charles Donnelly and Richard M. Stallman. Bison: the yacc-compatible parser generator. Technical report, Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, Tel: (617) 876-3296, 1988.
8. Margaret A. Ellis and Bjarne Stroustrup. *The Annotated C++ Reference Manual.* Addison-Wesley, Reading, Massachussetts, 1990.
9. Rodríguez-León Casiano. `Parse::Eyapp` Manuals. 2007.
   CPAN: `http://search.cpan.org/dist/Parse-Eyapp/`
   google-code: `http://code.google.com/p/parse-eyapp/`
10. Wall, L., Christiansen, T., Schwartz, R. (1996). *Programming Perl.* O'Reilly & Associates.
11. Allison Randal, Dan Sugalski, Leopold Totsch. *Perl 6 and Parrot Essentials.* O'Reilly Media. June 2004.
12. Hudak, P. Modular Domain Specific Languages and Tools. ICSR '98: Proceedings of the 5th International Conference on Software Reuse. IEEE Computer Society. Pages 134-142, June 1998.
13. ISO. Extended Pascal ISO 10206:1990.
    `http://www.standardpascal.org/iso10206.txt`.
14. Bryan Ford. Functional Pearl: Packrat Parsing: Simple, Powerful, Lazy, Linear Time.
    `http://pdos.csail.mit.edu/ baford/packrat/icfp02/packrat-icfp02.pdf`
    (2002).
15. Damian Conway. Regexp::Grammars. Add grammatical parsing features to Perl 5.10 regexes. `http://search.cpan.org/dist/Regexp-Grammars/`.

Using ontology in the development of domain-specific languages

Ines Čeh, Matej Črepinšek, Tomaž Kosar, Marjan Mernik

Faculty of electrical engineering and computer science, Smetanova 17,
2000 Maribor, Slovenia
{Ines.Ceh, Matej.Crepinsek, Tomaz.Kosar, Marjan.Mernik}@uni-mb.si

Abstract. Domain-specific languages (DSL) are programming languages devoted to solve problems in a specific domain. Development of a DSL includes the following phases: decision, analysis, design, implementation and deployment. The least known and examined are analysis and design. Although various formal methodologies exist, the domain analysis is still done informally, most of the time. A common reason why formal methodologies are not used as often as they could be is that they are very demanding. Instead of developing a new, less complex methodology, we propose that domain analysis could be replaced with a previously existing analysis in some other form. A particularly suitable form for such is ontology. This paper focuses on ontology based domain analysis and how it can be incorporated into the DSL design phase. We present preliminary results of the Ontology2DSL framework, which can be used to help transform ontology to DSL grammar.

Keywords: domain-specific language, domain analysis, ontology

1 Introduction

Programming languages are used for human-computer interaction. Depending on the purpose of their use, programming language can be divided into general-purpose languages (GPL) and DSL [1]. GPL, such as Java, C and C#, are designed to solve problems from any problem area. In contrast to GPLs, DSLs, such as Latex, SQL and BNF, are tailored to a specific application domain.

When developing new software a decision must be made as to which type of programming language will be used; GPL or DSL. The issue is further complicated if an appropriate DSL does not exist. Then, the decision is whether to start to develop with a GPL language or to start with the development of the required DSL and then develop the software system with it. Reasons for the use of DSL are as follows: easier programming, re-use of semantics, the easier verification and programmability for the end-users. However, DSL also have their disadvantages, for example high development costs. The key is to answer the question: "When to develop a DSL?" The simplest answer to this question is: a DSL should be developed whenever it is necessary to solve the problem, which belongs to a problem family and we expect that

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 185–196

in the future more problems from the same problem family will appear. A more detailed response can be found in [1].

DSL development consists of the following phases: decision, analysis, design, implementation and deployment [1]. DSL development phases are not equally researched. The least known and examined phases are the analysis and design.

The knowledge on the problem domain and its definition is achieved at the domain analysis phase. Various methodologies for domain analysis have been developed. Examples of such methodologies include: DSSA (Domain Specific Software Architectures) [2], FODA (Feature-Oriented Domain Analysis) [3], and ODM (Organization Domain Modeling) [4]. Often, formal methodologies are not used due to complexity and the domain analysis is done informally. This has the consequence of complicating future DSL development. Even if the domain analysis is done with a formal methodology, there aren't any clear guidelines how the output from domain analysis can be used in a language design process. The outputs of domain analysis consist of domain-specific terminology, concepts, commonalities and variabilities. Variabilities would have been entries in the design of DSL, while terminology and concepts should reflect in the DSL constructs, and commonalities could be incorporated into the executing DSL environment. Although it is known where the outputs of the domain analysis should be used, there is a need for clear instructions on how to make good use of the information, which are retrieved during the analysis phase, in the design stage of the DSL.

To partially solve aforementioned problems, we propose that domain analysis (hereinafter referred to as classic domain analysis (CDA)) can be performed with the use of existing techniques from other fields of computer science. A particularly suitable is ontology [5]. Ontology provides a vocabulary of a specialized domain. This vocabulary represents the domain objects, concepts and other entities. Some type of domain knowledge can be obtained from the relationships of the entities, presented by the vocabulary. Ontologies in the CDA have already been used in [6]. Whereas Tairas et al. apply ontology in the early stages of domain analysis to identify domain concepts; we propose that ontology replaces the CDA. They [6] also investigated how ontologies contribute to the design of the language. Ontologies in connection with DSL are also used by other authors. Guizzardi et al. [7] propose the usage of an upper ontology (top-level ontology) [8] to design and evaluate domain concepts. Walter et al. [9] apply ontologies to describe DSL. Bräuer and Lochmann [10] propose an upper ontology to describe interoperability among DSLs.

The proposed solution of the first problem, the use of ontologies, has a significant effect on the second problem related to CDA. It translates the problem »How to make good use of the information, retrieved during the analysis phase, in the design stage of the DSL?« into the problem »How to make good use of the information contained in an ontology in the design stage of DSL?« This paper focuses on ontology based domain analysis (OBDA) and how it can be incorporated into the DSL design phase. We present preliminary results of the Ontology2DSL framework, which can be used to help transform ontology to DSL grammar.

The organization of this paper is as follows. Section 2 is intended to represent the similarities and variabilities between the CDA and OBDA. Section 3 presents the development of grammar from ontology as well as the framework Ontology2DSL. The conclusion and future work are summarized in Section 4.

2 Domain analysis

2.1 Classic Domain Analysis (CDA)

The goal of CDA is to select and define the domain of focus and collect appropriate domain information and integrate them into a coherent domain model; the result of CDA [11]. A representation of the domain system properties and their dependencies is the domain model. The properties are either common or variable which is represented in the model along with the dependencies between the variable ones [11]. Beside the development of the domain model, CDA also includes domain planning, identification and scoping [11].

CDA can incorporate different methodologies. Methodologies differ based on the degree of formality, information extraction techniques or their products. We have listed the most known methodologies in the introduction. FODA has been proven as the most commonly used formal methodology in DSL development.

FODA is a CDA method that was developed by the Software Engineering Institute. It is known for its models and feature modeling. In FODA, feature is an end-user characteristic of a system. A FODA process consists of two phases: context analysis and domain modeling. The goal of context analysis is to determine the boundaries (scope) of the analyzed domain. The purpose of domain modeling is to develop a domain model. FODA domain modeling phase is comprised of the following steps: information analysis, features analysis and operational analysis. The main goal of information analysis is to capture domain knowledge in the form of domain entities and links between them. The result of information analysis is the information model. The result of feature analysis is feature model, which is presented below. Operational analysis results in the operational model. It represents how the application works and covers the links between objects in the informational model and the features in the feature model. An important product from the phase of domain modeling is the domain dictionary. It defines the terminology used in the domain and it also includes textual definitions of domain concepts and features.

A feature model consists of:

- *Feature diagram* (FD) represents a hierarchical decomposition of features and their kinds (mandatory, alternative, and optional feature). Mandatory features are features that each system must have in the domain. Alternative features are features of which a system can possess only one at a time. Optional features are features that system may or may not have. A system can also have more than one feature at a time. These features are called or-features. Features are also classified as atomic or composite. Whereas atomic features cannot be further subdivided in other features, composite features are defined in terms of other features. The root node of the diagram represents a concept and the remaining nodes represent features. An example of a feature diagram is shown in Fig. 1.
- *Feature definitions* describe all features (semantics).
- *Composition rules for features* describe which combinations are valid or invalid.

- *Rationale* for features represents reasons for choosing a feature.

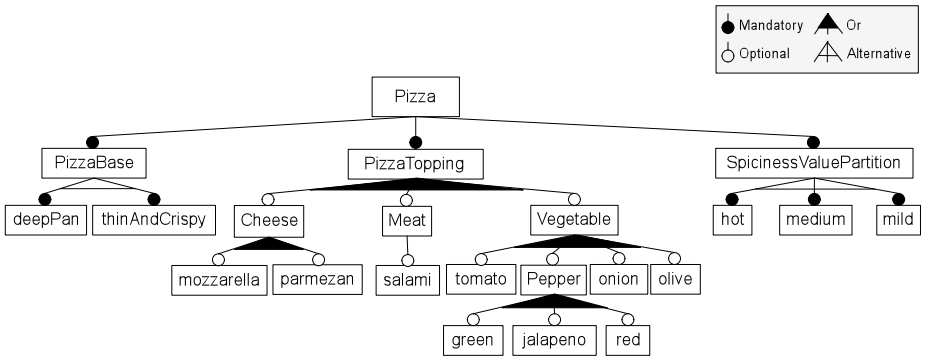


Fig. 1. Feature Diagram for a concept of a pizza.

Fig. 1 represents a simple FD of a pizza. The root node of the diagram, Pizza, represents a concept; the remaining nodes represent its features. Whereas mandatory features are indicated by a filled circle, optional features are indicated by an empty circle. Alternative and or-features are both indicated by a triangle, the former with an empty one and the latter with a filled triangle. The names of atomic features are written in lower-case while the composite features are written with their first letter in upper-case. Each pizza is composed of the pizza-base and at least one topping. The pizza-base is either "DeepPan" or "ThinAndCrispy", never both in the same pizza. The toppings are one or more of the following: "Cheese", "Meat" or "Vegetable". One pizza can have multiple toppings as well as multiple toppings of the same type (cheese topping of both "mozzarella" and "parmezan"). The pizza can be hot, medium or mild according to its spiciness (only one at the time).

Feature models are not only represented in the visual form of FDs but also in the textual form. Van Deursen and Klint [12] have proposed the feature description language (FDL) for the textual representation. The FDL definition constitutes of the feature definitions followed by a colon (":") and the features expression. Possible feature expression forms are presented in [12]. FDL exceeds the graphic feature diagram in the terms of expressive power and is appropriate for automatic processing. FD for pizza in FDL is listed below:

```
Pizza: all ( PizzaBase, PizzaTopping,
SpicinessValuePartition )
PizzaBase: one-of ( deepPan, thinAndCrispy )
PizzaTopping: more-of ( Cheese, Meat, Vegetable )
Cheese: more-of ( mozzarella, parmezan )
Meat: all ( salami? )
Vegetable: more-of ( tomato, Pepper, onion, olive )
Pepper: more-of ( green, jalapeno, red )
SpicinessValuePartition: one-of ( hot, medium, mild )
```

An important role of the FDs is to describe the variability of the programming system. The number of all possible configurations per system can be calculated with the use of variability rules, presented in [12].

Constraints, which are intended for variability reduction, are an optional component of the FDs. The constraints are enforced with the satisfaction rules [12]. The constraints are of two types [12]: diagram constraints and user constraints. The former include the "A1 requires A2" (if feature A1 is presented, then feature A2 should also be presented) and "A1 excludes A2" (if feature A1 is presented, then feature A2 should not be presented) constraints, while the latter include the "include A" (feature A should be present) and "exclude A" (feature A should not be present) constraints.

2.2 Ontology based Domain Analysis (OBDA)

There are many definitions of ontology in the literature and one of the most commonly used definitions is that of Gruber. He defined ontology as a "formal, explicit specification of shared conceptualization" [5]. Formal refers to the fact that it is machine readable. The specification is explicit because it summarizes the concepts, properties and relations between concepts. Furthermore, shared conceptualization contains knowledge that a group of experts has agreed upon. Conceptualization refers to the fact that it incorporates the target domain completely.

Ontologies are commonly encoded using ontology languages. Ontology languages can be divided in two major groups: traditional (i.e. Flogic, Ontolingua) and web-based languages (i.e. RDF(S), OWL, OWL 2) [13]. Recently, a new group of languages, rule-based (i.e. RuleML, SWRL), has emerged. These languages differ in their purpose and in their expressive power. The main requirements for an ontology language are: well defined syntax, well defined semantics, efficient reasoning support, sufficient expressive power and convenience of expression [14].

OWL is the most commonly used ontology language. It has three sublanguages; OWL Full, OWL DL and OWL Lite [14], [15]. These sublanguages have different levels of expressiveness. Whereas OWL Full is the most expressive, OWL Lite is the least expressive. Only OWL-DL allows automated reasoning.

The three components of OWL are: classes, properties, and individuals. Classes are interpreted as sets that contain individuals. Classes may be organized into a hierarchy. This means that a class can subsume other classes or it can be subsumed by other classes. The consequence of the subsumption relation is inheritance. Inheritance refers to the inheritance of properties which the children inherit from their parents. Whereas some ontologies only allow single inheritance, most ontologies, like OWL, allow multiple inheritance. OWL defines two special classes called „Thing" and „Nothing". Class Thing is the most general class and it is the superclass of every class that is included in ontology. Class Nothing is the empty and it is subclass of every included class. The class hierarchy for the truncated version of the previously existing Pizza ontology (PO) is shown in Fig. 2. The PO is used as an example in a practical guide to building OWL ontologies using Protégé [15]. PO has been choosen as an example because it includes the majority of the OWL features. The PO, written in OWL-DL, describes various pizzas based on their toppings.

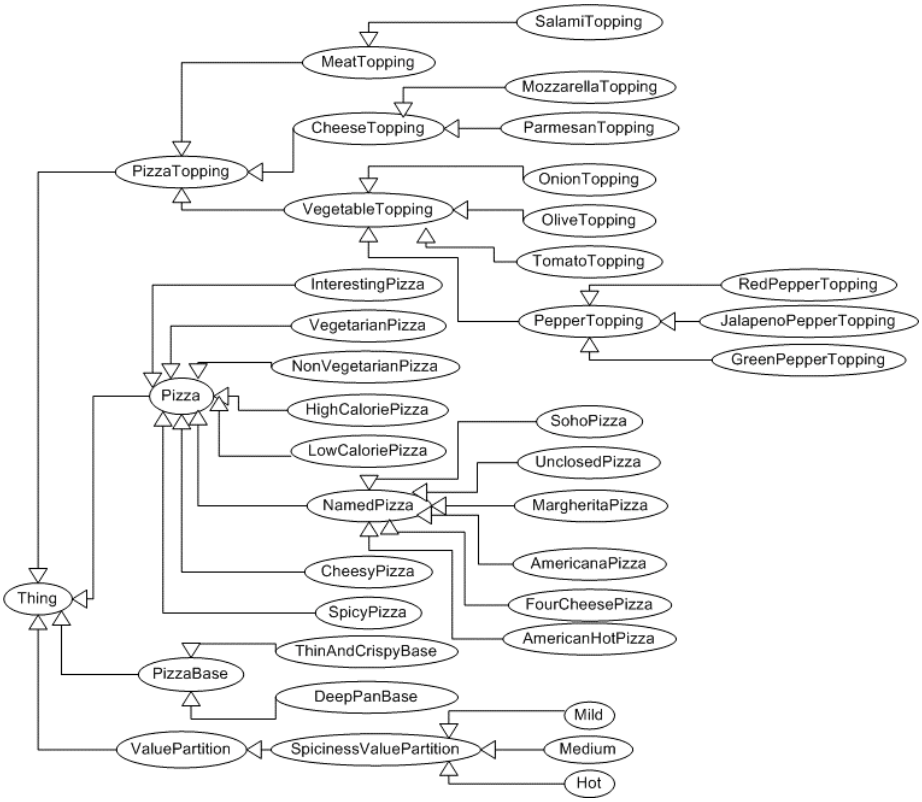


Fig. 2. Class hierarchy of Pizza Ontology.

Fig. 2 shows the class hierarchy of the PO used in this paper. The classes are represented with ellipses. All the classes are subclasses of the Thing class.

The second component, the properties, is a binary relation. OWL defines two main kinds of properties; object properties (i.e. hasTopping) and datatype properties (i.e. hasCaloricContentValue). Whereas object properties relate objects to other objects, datatype properties relate an object to datatype values. OWL supports XML schema primitive datatypes. The third component, the individuals, is the basic component of ontology. They represent objects in the domain of discourse. They can be concrete individuals (i.e. animals, airplanes, and people) as well as abstract individuals (i.e. words and numbers).

The relationships between classes are the means of the class definition in OWL. Such classes can be defined with the use of restrictions. Three main categories of restrictions that exist in OWL: quantifier restrictions (existential and universal), cardinality restrictions and „hasValue“ restrictions [15].

2.2 Comparison of CDA and OBDA

Both analysis incorporate a concept vocabulary, enable the display of property and class hierarchies, and provide a constraint mechanism. The CDA uses this mechanism for variability reduction while the OBDA uses it for the description of class properties. Both types of analysis describe semantics and are machine readable. The CDA differs from OBDA in its capability to record the reasons for the use of particular property (rationale) and the calculation of all possibilities. OBDA, on the other hand, provides the existence of objects, reasoning and querying. Numerous tools are available for it and ontologies are created across diverse research areas and are therefore available for use. The comparison shows that OBDA is capable of most of what the CDA is capable. The advantages of ontology are reasoning and querying, because they enable the validation of ontology. Valid ontology significantly reduces or prevents errors in DSL development. Semantics, that are inherently defined with the ontology, is also of great use when developing language semantics. Existing tools provide easy access to the ontology and enable efficient information extraction procedures. It is also a very important fact that ontologies are present in different research areas. That provides the method for elimination of domain analysis phase in DSL development and might significantly reduce the time needed for the language development.

Table 1. Comparison of CDA and OBDA

Property	FD + FDL	OWL ontology
Concept vocabulary	Features names	Name Class or property
Hierarchy	Feature diagram	Class hierarchy
Constraints	FDL constraints	Restrictions
Rationale	FD rationale properties	No
Objects	No	Individuals
Possible combinations	Variability rules (FDL)	No
Reasoning support	No	Reasoners (i.e. FaCT++)
Machine readable	Yes	Yes
Tools	In its infancy	Yes (i.e. Protege)
Semantics	Yes	Sets of relations
Domain analysis in use	No	Existing ontologies
Query support	No	Yes (DL Query)

The comparison leads to the conclusion that the CDA can indeed be replaced with OBDA, primarily because the ODBA provides everything needed for DSL development and adds new capabilities.

3 Language design

The grammar design is a feature of the Ontology2DSL framework. The framework enables the transformation of the OWL document to an appropriate internal data structure. The data structure is then transformed with the use of transformation patters. The resulting output is in the form of grammar and one or more programs. A DSL engineer that uses various tools at his/her disposal reviews them. If irregularities are found, they are resolved in accordance to their type in either the ontology or the transformational patterns. With regard to the type of the fix applied, the tool then uses new patterns on the old ontology, old patterns on the new ontology or new patterns on the new ontology. The process is repeated until the engineer finds no more irregularities, which finally results in the language grammar definition and one or more programs. The framework, besides the grammar development, can be supported with the development of DSL tools. They can be developed by the DSL engineer with the language development tools such as LISA [16]. Ontology2DSL framework is presented as a workflow diagram on Fig. 3.

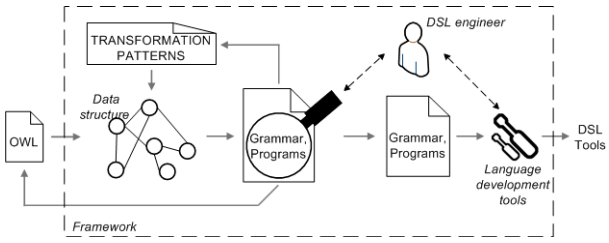


Fig. 3. Ontology2DSL framework.

3.1 Designing DSL grammar

Before starting with an explanation of basic steps of Ontology to DLS transformation (O2DSL), target ontology needs to be well understood. Language designer must understand what ontology describes and why it was designed. Moreover, language designer needs to know what are DSL requirements and what is the purpose of DSL. In most cases the DSL requirements and the ontology, do not overlap in all concepts.

In this paper, methodology O2DSL is demonstrated on a PO example. Single ontology, as well as PO example, can be used to develop many different DSLs. The purpose of the one presented here is a DSL to describe pizzas that can be queried by pizzas characteristics. Concepts of the simple pizza query language are not described

in the given PO, therefore are omitted from this example transformation. The DSL is named as Pizza Language (PL) and a result of O2DSL is the obtained language grammar [17]. As a side effect of transformation, domain data that is described in DSL programs is extracted.

Basic concept transformation. Ontology classes are interpreted as abstract groups and represent production rules in grammar non-terminals. In transformation classes, names have been used for grammar non-terminal names. Sequences and alternatives that describe production rules are obtained through class dependences and class hierarchy. For example, class hierarchy for `PizzaTopping` (Fig. 2) describes the group of toppings `CheeseTopping`, `MeatTopping` and `VegetableTopping`. Subgroup `CheeseTopping` includes subgroup of classes `ParmesanTopping` and `MozzarellaTopping`. In O2DSL transformation class hierarchy is transformed into production alternatives. Grammar example:

```
PizzaTopping ::=  CheeseTopping | MeatTopping |
                  VegetableTopping
CheeseTopping ::= ParmesanTopping | MozzarellaTopping
```

Generalization extraction. This pattern extracts abstract group from ontology and presents it as a configurable external element (program or constant attribute). In most cases this pattern is used for leaf classes in class hierarchy. For example in `PizzaTopping` (Fig. 2) leafs can be in general named `Topping`, that represents terminal in DSL grammar. Information of `ParmesanTopping` is then moved in DSL program as `Topping` instance (`parmesan`). For ontology class `CheeseTopping`, the following production is derived:

`CheeseTopping ::= Topping` and `Topping ::= #string`

Program fragment examples: `'parmesan'` and `'mozzarella'`.

Often, desired feature of DSL is its scalability. Therefore, the ability to define more instances of class `CheeseToping` is added to our DSL. The right side of `CheeseTopping` production is defined as a set of toppings:

```
CheeseTopping ::= {Topping}
```

Multi class generalization. It is common in the ontology that more than one class represents a similar domain concept, usually the only difference between these classes is in their derivation hierarchy. For example, in PO classes `ParmesanTopping`, `SalamiTopping`, `OnionTopping`, etc. they describe the same concept of topping, but they derive from different classes. In DSL grammar, this can be described with the same non-terminal. Grammar example:

```
CheeseTopping ::= Topping
MeatTopping ::= Topping
VegetableTopping ::= Topping
Topping ::= #string
```

Program fragment examples: `'parmesan'`, `'salami'` and `'onion'`.

Because of generalization we can get ambiguous grammar (one topping belongs to different abstract groups). To solve that problem, an enriching syntax can be used.

Enriching the syntax. One of the goals of DSL is to have a clear and easy to understand syntax with intuitive semantic. To achieve that, different patterns for enumerations, concepts that are seperated with brackets, adding reserved words, etc. have been used in O2DSL. For example in the `PizzaTopping` class reserved word `topping` is used. After all patterns are applyed, the following grammar is obtained:

```
PizzaTopping ::= topping
(CheeseTopping|MeatTopping|VegetableTopping)
CheeseTopping ::= cheese ToppingList
MeatTopping ::= meat ToppingList
VegetableTopping ::= vegetable ToppingList
ToppingList ::= '(' Topping {',' Topping} ')'
Topping ::= #string
```

Program fragment examples:

```
topping cheese ( 'parmesan' , 'mozzarella' )
topping meat ('salami')
```

In some cases, multi class generalization has an additional level in class hierarchy (additional abstract group between two classes). For example, `VegetableTopping` has an additional subclass `PepperTopping` that has subclasses `RedPepperTopping`, `GreenPepperTopping` and `JalapenoPepperTopping` (Fig. 2). One way to express an additional abstract group is to add a new alternative on `VegetableTopping` level with associated productions, other is by skipping this level and expect additional information in derivation of non-terminal `Topping`. Program fragment example:

```
topping vegetable ('red pepper', 'green pepper',
'jalapeno pepper')
```

Object properties and restriction transformations. Additional information about class relations can be obtained from ontology's object and class properties. For example `RedPepperTopping` has relations with `SpicinessValue-Partition` that has subclasses: `Hot`, `Mild` and `Medium`. In PO example, the relation is described by property `hasSpiciness Hot`. Class `PepperTopping` owns the property, therefore non-terminal `Topping` gets additional information. This property is not set for all `PizzaTopping` derivations and therefore it is optional. Following production is obtained:

```
Topping ::= #string [is SpicinessValuePartition]
```

Program fragment examples:

```
topping vegetable ('red pepper' is hot, 'green pepper' is
mild, 'jalapeno pepper' is medium, 'onion')
```

All class restrictions can be transformed in DSL by the similar transformation. In case that some restrictions are in addition defined by logical expression, support for logical expressions can also be added as part of DSL.

Obtained grammar. The appropriate order (sequence) of domain main concepts is defined from class hierarchy and class restrictions. For example, in case of PL, different instances of `PizzaTopping` must be defined before the definition of `NamedPizza`. Part of final PL grammar:

```
PL ::= {PizzaTopping} {Pizza} {Individual} {Query}
PizzaTopping ::= topping
(CheeseTopping|MeatTopping|VegetableTopping)
CheeseTopping ::= cheese ToppingList
MeatTopping ::= meat ToppingList
VegetableTopping ::= vegetable ToppingList
ToppingList ::= '(' Topping {',' Topping} ')'
Topping ::= #string [is SpicinessValuePartition]
SpicinessValuePartition ::= mild | hot | medium
Pizza ::= pizza (Interesting | Vegetarian | NonVegetarian
     | HighCalorie | LowCalorie | Named | Cheesy | Spicy)
…
```

Obtained DSL syntax is easy to understand and gives all flexibility and usability of DSLs. Obtained grammar and program fragments are used as a base for language development tool frameworks.

4 Conclusion and future work

In this paper, we have focused on the presentation of a new design methodology that enables the development of the language grammar, based on the OBDA. The limitations of the CDA have been examined and the replacement in the form of OBDA has been proposed. Both analysis have been presented and compared for similarities and differences. Grammar development, based on the OBDA, and the Ontology2DSL has also been briefly presented.

The results of the comparison between both analysis show that the OBDA is comparable to the CDA and also provides some additional information that can be used to specify language behavior. As such, it is also suitable as an alternative to CDA for grammar development. The framework Ontology2DSL is still under development. Currently, the framework supports the import of OWL ontology to an internal data structure and the transformation rules have been defined. The continuing development of the framework is a part of our future work. More specifically, we will focus on validation of the developed grammar and the use of previously unused information (i.e. for semantics development) that has been acquired with OBDA. The results of our research work will also be the transformation of the developed DSL to a form that is compatible with the compiler generators, such as LISA [16]. One of the future activities, to complete the methodology O2DSL, is evaluation of DSLs. As shown in the study [18], this activity is often underestimated by language developers. There is a plan to support this activity with tool based on questionnaire similar to [19] that will further improve the language.

References

1. Mernik, M., Heering, J., Sloane, A. M.: When and how to develop domain-specific languages. ACM Computing Surveys (CSUR) 37, 316--344 (2005)
2. Taylor, R. N., Tracz, W., Coglianese, L.: Software development using domain-specific software architectures. ACM SIGSOFT Software Engineering Notes 20, 27--38 (1995)
3. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA). Technical report, (1990)
4. Simos, M., Anthony, J.: Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering. In: Proceedings of the 5th International Conference on Software Reuse, pp. 94--102. IEEE Computer Society, (1998)
5. Gruber, T.R.: A translation approach to portable ontology specification. Knowledge Acquisition 5, 199--220 (0993)
6. Tairas, R., Mernik, M., Gray, J.: Using Ontologies in the Domain Analysis of Domain-Specific Languages. In: Models in Software Engineering. LNCS, vol. 5421, pp. 332--342. Springer, (2009)
7. Ontology-Based Evaluation and design of domain-specific visual modeling languages, http://www.loa-cnr.it/Guizzardi/ISD2005.pdf
8. Guarino, N.: Semantic Matching: Formal ontological distinctions for information organization, extraction, and integration. In: Information Extraction A Multidisciplinary Approach to an Emerging Information Technology. LNCS, vol. 1299, pp. 139--170. Springer, (1997)
9. Walter, T., Parreiras, F. S., Staab, S.: OntoDSL: An Ontology-Based Framework for Domain-Specific Languages. In: Model Driven Engineering Languages and Systems. LNCS, vol. 5795, pp. 408--422. Springer, (2009)
10. Bräuer, M., Lochmann, H.: An Ontology for Software Models and Its Practical Implications for Semantic Web Reasoning. In: The Semantic Web: Research and Applications. LNCS, vol. 5021, pp. 34--48. Springer, (2008)
11. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools and Applications. ACM Press/Addison-Wesley Publishing Co., (2000)
12. Van Deursen, A., Klint, P.: Domain-specific Language Design Requires Feature Descriptions. Journal of Computing and Information Technology 10, 1--17 (2002)
13. Corcho, Ó., Gómez-Pérez, A.: *A Roadmap to Ontology Specification Languages. In:* Knowledge Engineering and Knowledge Management Methods, Models, and Tools. LNCS, vol. 1937, pp. 80--96. Springer, (2000)
14. Antoniou, G., van Harmelen, F.: Handbook on Ontologies. Springer, Heidelberg (2009)
15. A Practical Guide to Building OWL Ontologies Using Protégé 4 and CO-ODE Tools, http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_2.pdf
16. Mernik, M., Lenič, M., Avdičaušević, E., Žumer, V.: LISA: An Interactive Environment for Programming Language Development. In: Horspool, N. (ed.) Compiler Construction. LNCS, vol. 2304, pp. 1-4. Springer, (2002)
17. Aho, A. V., Lam, M. S., Sethi, R., Ullman, J. D.: Compilers: Principles, Techniques, and Tools. Addison Wesley, (2007)
18. Gabriel, P., Goulão, M., Amaral, V.: Do Software Languages Engineers Evaluate their Languages? In: Proceedings of the XIII Congreso Iberoamericano en "Software Engineering" (CIbSE'2010), pp. 149--162. CIbSE2010 (*Ecuador*), (2010)
19. Haugen, O., Mohagheghi, P.: A Multi-dimensional Framework for Characterizing Domain Specific Languages. In: Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling (DSM'07), Montréal, Canada, (2007)

AGile, a structured editor, analyzer, metric evaluator, and transformer for Attribute Grammars

André Rocha, André Santos, Daniel Rocha, Hélder Silva, Jorge Mendes, José Freitas, Márcio Coelho, Miguel Regedor[2], Daniela da Cruz, and Pedro Rangel Henriques[1]

[1] University of Minho - Department of Computer Science,
Campus de Gualtar, 4715-057, Braga, Portugal
{danieladacruz,prh}@di.uminho.pt
[2] University of Minho - Department of Computer Science,
Master Course on Language and Grammar Engineering (1.st year)

Abstract. As edit, analyze, measure or transform attribute grammars by hand is an exhaustive task, it would be great if it could be automatized, specially for those who work in Language Engineering. However, currently there are no editors oriented to grammar development that cover all our needs.
In this paper we describe the architecture and the development stages of AGile, a structured editor, analyzer, metric calculator and transformer for attribute grammars. It is intended, with this tool, to fill the existing gap.
An AnTLR based attribute grammar syntax was used to define the input for this system. As soon as the user types the grammar, the input is parsed and kept in an intermediate structure in memory which holds the important information about the input grammar. This intermediate structure can be used to calculate all the metrics or to transform the input grammar.
This system can be a valorous tool for those who need to improve the performance or functionalities of their language processor, speeding up the difficult task of defining and managing a language. Features like highlighting, automatic indentation, on-the-fly error detection, etc., also adds efficiency.

1 Introduction

Editing an Attribute Grammar is a long and complex hard task. So it is important to provide basic support to make the editing easier. However, even more important is to assist the language engineering in designing and developing a language with quality. This requires that the grammar development environment (GDE) incorporates knowledge from grammar engineering, like analysis, visualization, metric evaluation, and transformation.

This article describes AGile, a text editor built to assist in the grammar writing process, comprising four main features as follows: *Syntax and lexical awareness*: syntax highlighting, automatic indentation, lexical integrity checking, syntactic integrity checking; *Metrics evaluation* for quality assessment (derivation Rules analysis: both size and structural parameters should be measured and metrics evaluated; attribute analysis: evaluate attribute-related metrics and other tasks — generate dependence graph, ...);

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 197–200

Derivation Rules transformation: transformations upon types of rules, namely eliminating unitary rules, eliminate left/right recursion, ...; *Visualization*: dependence graphs, for both the syntactic and the semantics components, will be built and displayed.

AGile GDE is a project under development in the context of a master course on Language Engineering. The motivation for this work is the application of all the theoretical concepts on grammar engineering (quality assessment and metrics) and on software analysis and transformation (analyzers, internal representations, transformers and visualizers).

AGile parses the input grammar to check its syntactic and semantics compliance with the meta-language defined (in our case, we have adopted AnTLR [1] meta-language syntax); if errors are detected during this phase, descriptive messages will be generated. Other important AGile components are the metrics evaluator, the grammar transformation and the visualizer. To support all these operations, the system generates, during parsing, an attributed abstract syntax tree (AST) and creates an Identifier Table (IdTab)). The operations referred above will be executed on this intermediate representation.

The paper has, after this, 5 sections. Section 2 describes the syntax-directed editor. Section 3 discusses the metrics evaluated by AGile. Section 4 introduces the rule transformations implemented by the GDE proposed. Section 5 very briefly present the visualization provided at moment. As usual, Section 6 closes the paper.

2 Editor

The first component of AGile is a text editor. Editors are tools that give us the ability to collect, prepare and arrange material for storing objects in a computer to be processed afterwards.

In our project the objects we want to edit are structured documents instead of unstructured text; actually, we need to process structured texts representing grammars. So we decided to developed an editor that could help to create and maintain that structure.

Structure text editors can be classified into different categories [2–4]: structure-aware editors; syntax-directed editors; language-based editors.

Our editor falls on the second category and it treats grammar rules in an analytical way, meaning that we are using an intermediate representation to offer the user the functionality described bellow.

To help in the process of writing a grammar, our editor uses the knowledge about the meta-grammar to offer the following set of features: line numbering — useful for debugging, allowing a quick identification of a line; syntax highlighting — the editor sends the text to the analyzer producing an intermediate structure obtained from the abstract syntax tree, then this structure is traversed producing tokens colored differently according to their lexical/syntactic role; automatic indention of the text according to the hierarchical relationship between components.

3 Metrics Evaluator

Among other factors that affect the software quality, its complexity has a large influence, and thus metrics were designed to assess the software complexity.

In the context of grammar quality, Power and Malloy have defined in [5] a set of metrics, derived from the above mentioned software metrics. In this paper, we do not follow strictly Power and Malloy's approach. We took there classification as a basis, but we go further, defining a set of 10 metrics classified in 3 types: size, shape and lexicography metrics.

One of the main components of AGile is the evaluation of size metrics on a context-free grammar, edited under this grammar processing environment. The first set of metrics is related with the size of the grammar G, and the second one is concerned with the size the Parser derived from the grammar G.

The set of metrics metrics evaluated by AGile are the following: **#T** — Number of terminal symbols; **#NT** — Number of non-terminal symbols; **#P** — Number of productions; **#PU** — Number of unitary productions; **$RHS**$_{avg}$ — Average size of the right hand side; **$Alt**$_{avg}$ — Average number of alternatives for each symbol; **Fan-in & Fan-out** — Number of dependencies between symbols from the dependence graph; **#TabLL(1)** — Size of the LL(1) parse table; **#RD** — Size of the Recursive Descendant parser.

In this way the user obtains easily and in an interactive mode immediate feedback about his grammar. Thus, the user is aware of the options that is doing when constructing the grammar.

4 Transformer

When we are writing a grammar for a certain language, the one that we wrote is not always the best when taking into account factors like: grammar metrics, grammar comprehension, or even the efficiency of the future parser. In this context, it is important to consider some transformation techniques in order to make our grammar a better one.

In AGile were introduced two different types of transformations. the *rename of the name* of a Terminal, a Non-Terminal, or Attributes; the *elimination of Unitary Productions*. The first transformation can increase significantly the clarity of the grammar, making easier the comprehension and maintenance tasks – it increases the grammar quality. When we have a long grammar and we want to change the name of a symbol that is mentioned in a lot of productions, it would be a time consuming and exhausting task to change all of them by hand. Automatizing it, reduces effort and at the end assures a complete replacement. The second transformation produces a grammar more difficult to understand, but the advantage is that from the reduced grammar (with less Productions and Non-Terminals) we can generate a more efficient Processor.

AGile computes automatically the metrics for the new grammar, allowing the user to compare both version (the original and the transformed).

5 Visualizer

To complement the comprehension of the grammar under development, this AGile component produces and displays a visual representation of a grammar. It is well known from the literature and practice that the visualization of dependence graphs offers an effective help for program comprehension. Extending the same idea to grammars, our GDE also offers a functionality to show the Dependence Graphs between Symbols, marking in each node its values of Fan-in and Fan-out.

6 Conclusion

This paper introduced AGile, a structured editor, analyzer, metric calculator and transformer for attribute grammars. Mainly, this tool allows the user to write a grammar, based on its specific syntax, and apply several actions over it.

Features like syntactic-directed edition, metrics evaluation and form transformations make AGile a useful tool for studying and formal grammars.

According to [6], grammar metrics have been introduced to measure the quality and complexity of a given grammar in order to direct the grammar engineering. Computing metrics since the early stages of the development of a given grammar, allows to improve the grammar and to avoid some undesirable features that otherwise would only be perceived later on.

Dependence graphs, for syntactic dependencies between grammar symbols or semantic ones between attributes, can also be generated and displayed to simplify the understanding of the grammar edited.

A lot of work still needs to be done to realize all our ideas, however the theoretical foundations are established and the prototype is promising.

References

1. Parr, T.: The Definitive ANTLR Reference: Building Domain-Specific Languages. First edn. Pragmatic Programmers. Pragmatic Bookshelf (Maio 2007)
2. Donzeau-Gouge, V., Huet, G., Kahn, G., Lang, B.: Programming environments based on structured editors: the mentor experience. Rapport de Recherche 26, INRIA, Rocquencourt (July 1980)
3. Teitelbaum, T., Reps, T.: The cornell program synthesizer: A syntax-directed programming environment. Communications of the ACM **24**(9) (Setembro 1981)
4. Reps, T.: Generating Language-Based Environments. PhD thesis, Cornell University (1982)
5. Power, J.F., Malloy, B.A.: A metrics suite for grammar-based software: Research articles. J. Softw. Maint. Evol. **16**(6) (2004) 405–426
6. Cervelle, J., Crepinsek, M., Forax, R., Kosar, T., Mernik, M., Roussel, G.: On defining quality based grammar metrics. In: Proceedings of the 2nd Workshop on Advances in Programming Languages (WAPL'2009), Mragowo, Poland, IEEE Computer Society Press (October 2009) 651–658

Efficient Retrieval of Subsumed Subgoals in Tabled Logic Programs

Flávio Cruz and Ricardo Rocha*

CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
{flavioc,ricroc}@dcc.fc.up.pt

Abstract. Tabling based systems use call similarity to decide when a tabled subgoal should produce or consume answers. Most tabling engines do that by using variant checks. A more refined method, named call subsumption, considers that a subgoal A will consume answers from a subgoal B if A is subsumed by B, thus allowing greater answer reuse. Recently, we have developed an extension, called *Retroactive Call Subsumption*, that improves upon call subsumption by supporting bidirectional sharing of answers between subsumed/subsuming subgoals. In this paper, we present an algorithm to efficiently retrieve the set of currently evaluating subgoals that are subsumed by a more general subgoal.

1 Introduction

Tabled resolution methods solve some of the shortcomings of Prolog because they can considerably reduce the search space, avoid looping and have better termination properties than SLD resolution based methods [1]. Tabling works by memorizing generated answers and then by reusing them on *similar calls* that appear during the resolution process. In a nutshell, first calls to tabled subgoals are considered *generators* and are evaluated as usual, using SLD resolution, but their answers are stored in a global data space, called the *table space*. Similar calls are called *consumers* and are resolved by consuming the answers already stored for the corresponding generator, instead of re-evaluating them against the program clauses. There are two main approaches to determine if a subgoal A is similar to a subgoal B: *call variance* and *call subsumption*.

In call variance, A and B are similar if they can be identical through variable renaming. For example, $p(X, 1, Y)$ and $p(Y, 1, Z)$ are *variants* because both can be transformed into $p(VAR_0, 1, VAR_1)$. Tabling by call subsumption is based on the principle that if A is subsumed by B (i.e., if A is an instance or more specific than B) and S_A and S_B are the respective answer sets, therefore $S_A \subseteq S_B$. For example, subgoal $p(X, 1, 2)$ is subsumed by subgoal $p(Y, 1, Z)$ because there is a substitution $\{Y = X, Z = 2\}$ that makes $p(X, 1, 2)$ an instance of $p(Y, 1, Z)$. For some types of programs, call subsumption yields superior time performance, as it

* This work has been partially supported by the FCT research projects STAMPA (PTDC/EIA/67738/2006) and HORUS (PTDC/EIA-EIA/100897/2008).

allows greater reuse of answers, and better space usage, since the answer sets for the subsumed subgoals are not stored. Arguably, the most successful approach for subsumption-based tabling is the *TST (Time-Stamped Trie)* design [2].

Despite the advantages of using subsumption-based tabling, the degree of answer reuse depends on the call order of subgoals. If a more general subgoal is called before specific subgoals, answer reuse will happen, but if more specific subgoals are called before a more general subgoal, no reuse will occur. To solve this problem, we implemented an extension to the original TST design, called *Retroactive Call Subsumption (RCS)* [3], that supports subsumption-based tabling by allowing full sharing of answers between subsumptive subgoals, independently of the order they are called. RCS works by selectively pruning the evaluation of subsumed subgoals when a more general subgoal appears later on. In this paper, we describe the modifications made to the table space data structures and we discuss the new algorithm developed to efficiently retrieve the set of currently evaluating instances of a subgoal.

2 Retrieval of Subsumed Subgoals

2.1 Table Space Data Structures

Arguably, the most successful data structure for representing the table space is *tries* [4]. Tries are trees in which common prefixes are represented only once. Tries provide complete discrimination for terms and permit lookup and insertion to be done in a single pass. In a trie-based tabling system, each tabled predicate has a *table entry* that points to a *subgoal trie*. In the subgoal trie, each distinct trie path represents a tabled subgoal call and each leaf trie node points to a *subgoal frame*, a data structure containing information about the subgoal call.

In our new approach, each subgoal trie node was extended with a new field, named *in_eval*, which stores the number of subgoals, represented below the node, that are in evaluation. This field is used to, during the search for subsumed subgoals, prune the subgoal trie branches without evaluating subgoals, i.e., the ones with $in\_eval = 0$.

When a subgoal starts being evaluated, all subgoal trie nodes in its subgoal trie path get the *in_eval* field incremented. When a subgoal completes its evaluation, the path is decremented. Hence, for each subgoal leaf trie node, the *in_eval* field can be equal to either: 1, when the corresponding subgoal is in evaluation; or 0, when the subgoal is completed. For the root subgoal trie node, we know that it will always contain the total number of subgoals being currently evaluated.

When a chain of sibling nodes is organized in a linked list, it is easy to select the trie branches with evaluating subgoals by looking for the nodes with $in\_eval > 0$. But, when the sibling nodes are organized in an hash table, it can become very slow to inspect each node as the number of siblings increase. In order to solve this problem, we designed a new data structure, called *evaluation index*, in a similar manner to the *time stamp index* [2] of the TST design. An evaluation index is a double linked list that is built for each hash table and is

used to chain the subgoal trie nodes where the *in_eval* field is greater than 0. The evaluation index makes the operation of pruning trie branches much more efficient by providing direct access to trie nodes with evaluating subgoals. While advantageous, the operation of incrementing or decrementing a subgoal trie path is more costly, because these indexes must be maintained.

2.2 Matching Algorithm

The algorithm that finds the currently running subgoals that are subsumed by a more general subgoal S works by matching the subgoal arguments SA of S against the trie symbols in the subgoal trie T. By using the *in_eval* field as described previously, we can prune irrelevant branches as we descend the trie. When reaching a leaf node, we append the corresponding subgoal frame in a result list that is returned once the process finishes. If the matching process fails at some point or if a leaf node was reached, the algorithm backtracks to try alternative branches, in order to fully explore the subgoal trie T.

When traversing T, trie variables cannot be matched against ground terms of SA. Ground terms of SA can only be matched with ground terms of T. For example, if matching the trie subgoal $p(VAR_0, VAR_1)$ with the subgoal $p(2, X)$, we cannot match the constant 2 against the trie variable VAR_0, because $p(2, X)$ does not subsume $p(VAR_0, VAR_1)$.

When a variable of SA is matched against a ground term of T, subsequent occurrences of the same variable must also match the same term. As an example, consider the trie subgoal $p(2, 4)$ and the subgoal $p(X, X)$. The variable X is first matched against 2, but the second matching, against 4, must fail because X is already bound to 2.

Now consider the trie subgoal $p(VAR_0, VAR_1)$ and the subgoal $p(X, X)$. Variable X is first matched against VAR_0, but then we have a second match against a different trie variable, VAR_1. Again, the process must fail because $p(X, X)$ does not subsume $p(VAR_0, VAR_1)$. This last example evokes a new rule for variable matching. When a variable of SA is matched against a trie variable, subsequent occurrences of the same variable must always match the same trie variable. This is necessary, because the found subgoals must be *instances* of S. Therefore, this problem can be reduced to the task of finding all instances of S in trie T. To implement this algorithm, we use the following data structures:

- *WAM data structures*: we take advantage of the existent Prolog data structures based on WAM machinery: heap, trail, and associated registers. The heap is used to build structured terms, in which the subgoal arguments are bound. Whenever a new variable is bound, we trail it using the WAM trail;
- *term stack*: stores the remaining terms to be matched against the subgoal trie symbols;
- *term log stack*: stores already matched terms from the term stack and is used to restore the state of the term stack when backtracking;
- *variable enumerator vector*: used to mark the term variables that were matched against trie variables;

– *choice point stack*: stores choice point frames, where each frame contains information needed to restore the computation in order to search for alternative branches.

The procedure that traverses a subgoal trie and collects the set of subsumed subgoals of a given subgoal call can be summarized in the following steps:

1. setup WAM machinery and push subgoal arguments into the term stack.
2. fetch a term T from the term stack;
3. search for a trie node N where the *in_eval* field is not 0.
4. search for the next node with a valid *in_eval* field to be pushed on the choice point stack, if any;
5. match T against the trie symbol of N;
6. proceed into the child of N or, if steps 3 or 5 fail, backtrack by popping a frame from the choice point stack and use the alternative trie node;
7. once a leaf is reached, add the corresponding subgoal frame to the resulting subgoal frame list. If there are choice points available, backtrack to try them;
8. if no more choice point frames exist, return the found subsumed subgoals.

3 Conclusions

We presented a new algorithm for the efficient retrieval of subsumed subgoals in tabled logic programs. Our proposal takes advantage of the existent WAM machinery and data areas and extends the subgoal trie data structure with information about the evaluation status of the subgoals in a branch, which allows us to prune the search space considerably. We therefore argue that our approach can be easily ported to other tabling engines, as long they are based on WAM technology and use tries for the table space.

Initial experiments using the YapTab tabling engine with support for retroactive call subsumption, showed low overheads on programs that do not benefit from the new algorithm, when compared to traditional call subsumption, and very good results when applied to programs that take advantage of it [3].

References

1. Chen, W., Warren, D.S.: Tabled Evaluation with Delaying for General Logic Programs. Journal of the ACM **43**(1) (1996) 20–74
2. Johnson, E., Ramakrishnan, C.R., Ramakrishnan, I.V., Rao, P.: A Space Efficient Engine for Subsumption-Based Tabled Evaluation of Logic Programs. In: Fuji International Symposium on Functional and Logic Programming. Number 1722 in LNCS, Springer-Verlag (1999) 284–300
3. Cruz, F., Rocha, R.: Retroactive Subsumption-Based Tabled Evaluation of Logic Programs. In: European Conference on Logics in Artificial Intelligence. LNCS, Springer-Verlag (2010) To appear.
4. Ramakrishnan, I.V., Rao, P., Sagonas, K., Swift, T., Warren, D.S.: Efficient Access Mechanisms for Tabled Logic Programs. Journal of Logic Programming **38**(1) (1999) 31–54

Mixed-Strategies for Linear Tabling in Prolog

Miguel Areias and Ricardo Rocha*

CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
{miguel-areias,ricroc}@dcc.fc.up.pt

Abstract. Tabling is an implementation technique that solves some limitations of Prolog's operational semantics in dealing with recursion and redundant sub-computations. Arguably, the SLDT and DRA strategies are the two most successful extensions to standard linear tabled evaluation. In this work, we propose a new strategy for linear tabling, named DRS, and we present a framework, on top of the Yap system, that supports the combination of variants of these three strategies.

1 Introduction

Tabled evaluation is a recognized and powerful technique that can considerably reduce the search space, avoid looping and have better termination properties than SLD resolution [1]. Tabling consists of storing intermediate solutions for subgoals so that they can be reused when a repeated subgoal appears during the resolution process. We can distinguish two main categories of tabling mechanisms: *suspension-based tabling* and *linear tabling*.

Suspension-based tabling mechanisms need to preserve the computation state of suspended tabled subgoals in order to ensure that all solutions are correctly computed. Linear tabling mechanisms use iterative computations of tabled subgoals to compute fix-points. While suspension-based mechanisms are considered to obtain better results in general, they have more memory space requirements and are more complex and hard to implement than linear tabling mechanisms.

Arguably, the SLDT [2] and DRA [3] strategies are the two most successful extensions to standard linear tabling evaluation. As these strategies optimize different aspects of the evaluation, they are, in principle, orthogonal to each other and thus it should be possible to combine both in the same system. In this work, we propose a new strategy, named *Dynamic Reordering of Solutions (DRS)*, and we present a framework, on top of the Yap Prolog system, that integrates and supports the combination of these three strategies. Our implementation shares the underlying execution environment and most of the data structures used to implement tabling in Yap. We thus argue that all these common support features allow us to make a first and fair comparison between these different linear tabling strategies and, therefore, better understand the advantages and weaknesses of each, when used solely or combined with the others.

* This work has been partially supported by the FCT research projects STAMPA (PTDC/EIA/67738/2006) and HORUS (PTDC/EIA-EIA/100897/2008).

2 Standard Linear Tabled Evaluation

Tabling works by storing intermediate solutions for tabled subgoals so that they can be reused when a repeated call appears. In a nutshell, first calls to tabled subgoals are considered *generators* and are evaluated as usual, using program resolution, but their solutions are stored in a global data space, called the *table space*. Repeated calls to tabled subgoals are considered *consumers* and are not re-evaluated against the program clauses because they can potentially lead to infinite loops, instead they are resolved by consuming the solutions already stored for the corresponding generator. During this process, as further new solutions are found, we need to ensure that they will be consumed by all the consumers, as otherwise we may miss parts of the computation and not fully explore the search space. To do that, linear tabling mechanisms maintain a single execution tree where tabled subgoals are iteratively computed until reaching a fix-point.

A generator call C thus keeps trying its matching clauses until reaching a fix-point. If no new solutions are found during one cycle of trying the matching clauses, then we have reached a fix-point and we can say that C is completely evaluated. However, if a number of subgoal calls is mutually dependent, thus forming a *Strongly Connected Component (SCC)*, then completion is more complex and we can only complete the calls in a SCC together. SCCs are usually represented by the *leader call*, i.e., the generator call which does not depend on older generators. A leader call defines the next completion point, i.e., if no new solutions are found during one cycle of trying the matching clauses for the leader call, then we have reached a fix-point and we can say that all subgoal calls in the SCC are completely evaluated.

3 Linear Tabling Strategies

The standard linear tabling mechanism uses a naive approach to evaluate tabled logic programs. Every time a new solution is found during the last round of evaluation, the complete search space for the current SCC is scheduled for re-evaluation. However, some branches of the SCC can be avoided, since it is possible to know beforehand that they will only lead to repeated computations, hence not finding any new solutions. Next, we present three different approaches for optimizing standard linear tabled evaluation.

3.1 Dynamic Reordering of Execution

The first optimization, that we call *Dynamic Reordering of Execution (DRE)*, is based on the original SLDT strategy, as proposed by Zhou et al. [2]. The key idea of the DRE strategy is to let repeated calls to tabled subgoals execute from the *backtracking clause of the former call*. A first call to a tabled subgoal is called a *pioneer* and repeated calls are called *followers* of the pioneer. When backtracking to a pioneer or a follower, we use the same strategy, first we explore the remaining clauses and only then we try to consume solutions. The fix-point check operation is still only performed by pioneer calls.

3.2 Dynamic Reordering of Alternatives

The key idea of the *Dynamic Reordering of Alternatives (DRA)* strategy, as originally proposed by Guo and Gupta [3], is to memorize the clauses (or alternatives) leading to consumer calls, the *looping alternatives*, in such a way that when scheduling an SCC for re-evaluation, instead of trying the full set of matching clauses, we only try the looping alternatives. Initially, a generator call C explores the matching clauses as in standard evaluation and, if a consumer call is found, the current clause for C is memorized as a looping alternative. After exploring all the matching clauses, C enters the *looping state* and from this point on, it only tries the looping alternatives until reaching a fix-point.

3.3 Dynamic Reordering of Solutions

The last optimization, that we named *Dynamic Reordering of Solutions (DRS)*, is a new proposal that can be seen as a variant of the DRA strategy, but applied to the consumption of solutions. The key idea of the DRS strategy is to memorize the solutions leading to consumer calls, the *looping solutions*. When a non-leader generator call C consumes solutions to propagate them to the context of the previous call, if a consumer call is found, the current solution for C is memorized as a looping solution. Later, if C is scheduled for re-evaluation, instead of trying the full set of solutions, it only tries the looping solutions plus the new solutions found during the current round. In each round, the new solutions leading to consumer calls are added to the previous set of looping solutions.

4 Experimental Results

To the best of our knowledge, Yap is now the first tabling engine that integrates and supports the combination of different linear tabling strategies. We have thus the conditions to better understand the advantages and weaknesses of each strategy when used solely or combined with the others. In what follows, we present initial experiments comparing linear tabled evaluation with and without support for the DRE, DRA and DRS strategies. To put the performance results in perspective, we used the well-known $path/2$ predicate, that computes the transitive closure in a graph, combined with several different graph configurations.

Next, we show in Table 1 the execution time ratios of standard linear tabled evaluation to DRE, DRA and DRS solely and combined strategies. Ratios higher than 1.00 mean that the respective strategies have a positive impact on the execution time. The results obtained are the average of 5 runs for each configuration.

Globally, the results in Table 1 show that, for most of these experiments, DRE evaluation has no significant impact in the execution time. On the other hand, the results indicate that the DRA and DRS strategies are able to effectively reduce the execution time for most of the experiments, when compared with standard evaluation, and that by combining both strategies it is possible to reduce even further the execution time of the evaluation. In most cases, this

Table 1. Execution time ratios of standard to DRE, DRA and DRS strategies

Strategy	Pyramid			Cycle			Grid		
	1000	**2000**	**3000**	**1000**	**2000**	**3000**	**20**	**30**	**40**
DRE	0.98	1.00	0.88	0.94	0.95	**1.04**	0.83	0.99	0.99
DRA	**1.60**	**1.59**	**1.58**	**1.18**	**1.20**	**1.22**	**1.08**	**1.09**	**1.07**
DRS	0.99	0.98	0.99	**1.14**	**1.18**	**1.25**	**1.20**	**1.20**	**1.21**
DRE+DRA	1.58	**1.66**	**1.63**	**1.22**	**1.24**	1.22	**1.12**	**1.10**	1.07
DRE+DRS	1.00	**1.01**	**1.01**	**1.22**	**1.23**	1.23	0.95	1.14	1.14
DRA+DRS	**1.63**	**1.64**	**1.62**	**1.56**	**1.59**	**1.69**	**1.40**	**1.32**	**1.32**
DRE+DRA+DRS	1.59	1.57	1.56	**1.61**	1.55	1.60	1.36	**1.33**	1.30

reduction is higher than the sum of the reductions obtained with each strategy individually. This shows the potential of our framework and suggests that the overhead associated with this combination is negligible. When DRE is present, the results are, in general, worst than the results obtained with the DRA/DRS strategies solely. A possible explanation for this behavior is the fact that, as DRE has more space requirements, this leads to more expansions of the execution stacks, which in turn can lead to higher ratios of cache and page misses. Still, these results require further study and analysis.

5 Conclusions

We have presented a new strategy for linear tabled evaluation of logic programs, named DRS, and a framework, on top of the Yap system, that integrates and supports the combination of different linear tabling strategies. Our experiments for DRS evaluation showed that, the strategy of avoiding the consumption of non-looping solutions in re-evaluation rounds, can be quite effective for programs that can benefit from it, with insignificant costs for the other programs. Preliminary results for the combined framework were also very promising. In particular, the combination of the DRA and DRS strategies showed the potential of our framework to reduce even further the execution time of a linear tabled evaluation.

References

1. Chen, W., Warren, D.S.: Tabled Evaluation with Delaying for General Logic Programs. Journal of the ACM **43**(1) (1996) 20–74
2. Zhou, N.F., Shen, Y.D., Yuan, L.Y., You, J.H.: Implementation of a Linear Tabling Mechanism. In: Practical Aspects of Declarative Languages. Number 1753 in LNCS, Springer-Verlag (2000) 109–123
3. Guo, H.F., Gupta, G.: A Simple Scheme for Implementing Tabled Logic Programming Systems Based on Dynamic Reordering of Alternatives. In: International Conference on Logic Programming. Number 2237 in LNCS, Springer-Verlag (2001) 181–196

Parser Generation in Perl: an Overview and Available Tools

Hugo Areias[1], Alberto Simões[2], Pedro Henriques[1], and Daniela da Cruz[1]

[1]Departamento de Informática, Universidade do Minho
[2]Escola Superior de Estudos Industriais e de Gestão, Instituto Politécnico do Porto
hugomsareias@gmail.com,pedrorangelhenriques@gmail.com
alberto.simoes@eu.ipp.pt,danieladacruz@di.uminho.pt

Abstract. There are some modules on Comprehensive Perl Archive Network to help with the parser generation process in Perl. Unfortunately, some are limited, only supporting a particular parser algorithm and do not covering some developer needs. In this document we will analyse some of these modules, testing them in terms of performance and usability, allowing a proper evaluation of the results and careful considerations about the state of art of parser generation using Perl.

Keywords: Parser generators, Perl, grammars

1 Introduction

The primary aim of this paper is to provide an overview of the particular condition of parser generation in Perl and analyse some of the available tools.
In the Comprehensive Perl Archive Network (CPAN[1]) there are some modules available to automate the process of generating a parser. However, the user should choose carefully according to his needs and because the lack of maintenance and efficiency of some of them.
We chose four tools, the most used, the more robust, the more elaborate and the more recent:

- **Parse::RecDescent** (v 1.962.2) – one of the most used tools, generate on-the-fly a recursive-descent parser;
- **Parse::Yapp** (v 1.05) – can be compared with the well known `yacc` parser generator tool in terms of algorithm and syntax;
- **Parse::Eyapp** (v 1.154) – an extended version of `Parse::Yapp` including new recursive constructs;
- **Regexp::Grammars** (v 1.002) – an implementation of the future Perl 6 grammars[2]

[1] http://www.cpan.org/
[2] This tool is only supported in recent Perl versions (> 5.10).

1.1 Parse::RecDescent

`Parse::RecDescent` [3] supports $LL(1)$ parsers [1] and generates recursive-descent parsers on-the-fly. It is a powerful module that provides useful mechanisms to create parsers with ease, such as auto-actions (automatically adding pre-defined actions) and named access to semantic rule values (allowing the retrieve of data from an associative array using the symbol name instead of the usual array indexes). To create the parser, `Parse::RecDescent` generates routines in runtime, doing the lexical and syntactic analysis, and achieving the results on the fly. The drawbacks are the incapacity to deal with left recursion and its efficiency when dealing with large inputs. Therefore, it is not recommendable for cases where the performance is an issue.

1.2 Parse::Yapp

`Parse::Yapp` [5] is one of the oldest parser generators in Perl and probably still one of the most robust. It is based on `yacc` [2]. Just like `yacc`, it is well known for supporting $LALR$ parsers [1] and for its parsing speed. Such traits makes it an obvious choice for the users. As an addition, it also provides a command-line script that, when executed over an input grammar file, generates a Perl Object Oriented (OO) parser. This module only supports *Backus-Naur Form* (BNF) rules to write the grammar. Also, `Parse::Yapp` does not include lexical analyser features, forcing the user to provide one. Gratefully, there are some useful modules on CPAN to help in this process, such as Text::RewriteRules [8].

1.3 Parse::Eyapp

`Parse::Eyapp` [6] is an extension of `Parse::Yapp`. Just like `yapp`, it only supports $LALR$ parsers, but is able to parse extended BFN rules. While it introduces a lot of new useful features, it still keeps the same structure of `Parse::Yapp` allowing parsers made for the second to run when executed by the first. The most relevant features from `Parse::RecDescent` implemented in this module include auto-actions and named access to semantic rule values.

1.4 Regexp::Grammars

`Regexp::Grammars` [4] is a module that tries to implement Perl 6 grammar support with Perl 5. This is possible given the new recursive regular expressions introduced in Perl 5.10. The module extends the regular expressions in a way that makes them similar to typical grammars. While it is easy to use, it has some efficiency problems, very similar to the presented for `Parse::RecDescent`, given that it also generates recursive-descent parsers. Also, `Regexp::Grammars` creates automatically abstract data structures for the grammar, reducing the number of visible semantic actions.

2 Analysis and Tests

Three different grammars[3] were chosen to help testing the four modules described earlier: The Swedish Chef, a simple grammar but relatively large with an high number of semantic actions; The Lavanda, a Domain Specific Language (DSL) to describe the laundry bags daily sent to wash by a launderette company; and an highly recursive grammar to match s-expressions. These tests were performed by a machine with an *Intel Pentium 4* with a clock rate of 3.4 GHz and 3Gb of *RAM*.
Looking to the following tables it is possible to understand the most efficient modules. `Parse::RecDescent` and `Regexp::Grammars` both use regular expressions to perform the lexical analysis but they store the parsing functions in memory as they are generated on-the-fly. So, even with the advantages of using regular expressions, these modules take too long. This also has to do with a few recursive-descent parser limitations in Perl.

Table 1. User time evolution of the four approaches for the Lavanda grammar.

Nr. Lines	Parse::Yapp	Parse::Eyapp	Parse::RecDescent	Regexp::Grammars
10	0.031 s	0.090 s	0.123 s	0.069 s
100	0.115 s	0.184 s	0.258 s	0.163 s
1000	1.240 s	1.380 s	4.041 s	1.399 s
10000	34.896 s	37.640 s	331.814 s	out of memory
100000	> 2488.348 s	> 4973.639 s		
1000000				

Table 2. Memory consumption (in megabytes) of the four approaches for the Lavanda grammar.

Nr Lines	Parse::Yapp	Parse::Eyapp	Parse::RecDescent	Regexp::Grammars
10	0.933	3.866	3.583	3.490
100	1.934	4.867	4.607	22.545
1000	12.141	15.214	15.175	181.809
10000	108.697	113.242	115.383	out of memory
100000				
1000000				

Table 3 show a final analysis of the modules. From these results, it is easy to realise that `Parse::Yapp` is the most efficient module available for Perl, mainly due to the fact that it is based on *LALR* grammars, slightly more powerful than *LL* algorithms. It also offers the best support for integration of the parser with other code. In the other hand, it does not offer any support for attribute grammars and for the construction of AST. The lack of documentation makes it not very easy to start with, increasing the development time. Also, it does not provides the best support for semantic actions when compared to the other modules and it requires the lexical analyser to be provided by the user.

[3] All grammars, test files and generated parsers are available at `http://www.di.uminho.pt/~gepl/PARSINGinPERL`

Table 3. Module Analysis.

Module	Debugging	Generated Parser Readability	Integration with External Code	Development Time
Parse::Yapp	+/−	+/−	++	+/−
Parse::Eyapp	+/−	+/−	+	+/−
Parse::RecDescent	+	NA	+	−
Regexp::Grammars	++	NA	+/−	−−

3 Conclusions

Parser generators in Perl still lacks valuable mechanisms to make them challengeable when compared with other languages, like C. There is no valid support for attribute grammars and, according to the research made, there is only one module on CPAN that supports attribute grammars that, however, lacks of maintenance for several years now.
The modules that support recursive-descent parsers provide several useful mechanisms but due to the lack of efficiency, they are not recommendable for processing large input streams. *LALR* parsers provide a more efficient solution, however the lexical analyser must be provided by the user and their efficiency is not the best when compared with other language solutions [7].
An alternative solution could be combining the Perl modules with other tools written in another languages to achieve better results. This solution would require a bridge between both tools and its evaluation would be dependable on the effort and difficulty level of implementing this bridge. This is precisely the objective of a master thesis that aims at retargeting AnTLR (a well known LL(K) compiler generator from attribute grammars) to generate Perl compilers.

References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers Principles, Techniques and Tools.* Addison-Wesley, 1986.
2. Stephen Johnson Bell and Stephen C. Johnson. Yacc: Yet another compiler-compiler. Technical report, 1979.
3. Damian Conway. Parse::recdescent. `http://search.cpan.org/dist/Parse-RecDescent/lib/Parse/RecDescent.pm`, 1997.
4. Damian Conway. Regexp::grammars. `http://search.cpan.org/~dconway/Regexp-Grammars-1.001005/lib/Regexp/Grammars.pm`, 2009.
5. Francois Desarmenien. Parse::yapp. `http://search.cpan.org/dist/Parse-Yapp/lib/Parse/Yapp.pm`, 1998.
6. Casiano Rodriguez-Leon. Parse::eyapp. `http://search.cpan.org/dist/Parse-Eyapp/lib/Parse/Eyapp.pod`, 2006.
7. Alberto Simoes. Parsing with perl. Copenhaga, Aug 2008. Yet Another Perl Conference Europe.
8. Alberto Simoes and José Joao Almeida. Text::rewriterules. `http://search.cpan.org/~ambs/Text-RewriteRules-0.21/lib/Text/RewriteRules.pm`, 2004.

Realizing Bidirectional Transformations in Attribute Grammars

João Saraiva[1] and Eric Van Wyk[2]

[1] University of Minho, Braga, Portugal
[2] University of Minnesota, Minneapolis, Minnesota, USA

Abstract. This position paper considers the possibility of implementing bidirectional transformations in modern attribute grammar systems. Such systems support features such as higher-order attributes, reference attributes, and forwarding of attribute queries. In the 1980's Yellin considered bidirectional transformations in attribute grammars lacking these features. But these features may open the door to automatically generating more expressive and powerful bidirectional transformations; this paper discusses the prospects of doing so.

1 Introduction

Interest in bidirectional transformations has increased in the past few years and has been studied in a number of computing disciplines [2]. In the bidirectional transformation literature the function that maps the source to the target, also called the view, is called the forward or "*get*" transformation. The function mapping the target back to the source is called the backward or "*put*" transformation. Of special interest is computing the *put* transformation after some modification of the target; this is illustrated in Figure 1.

The most interesting bidirectional transformations are those in which the source language is the richer of the two languages. Consider, for example, transformations between the concrete (source) and abstract (target) context free grammars of arithmetic expressions. The source language is the concrete syntax; the additional richness of the source (realized as more nonterminals and productions than in the target) is used to express the precedence and association of the infix binary operators. In many cases, a related collection of concrete nonterminals map to a single nonterminal in the abstract syntax. It is also often the case that several distinct source language constructs map to the same construct in the target. In the arithmetic expression example we may treat the unary negation of an expression e in the source as syntactic sugar for $0 - e$ in the target. Thus the source concrete subtraction and unary negation both map to subtraction in the target abstract syntax.

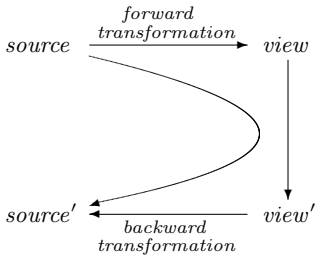


Fig. 1. Bidirectional Transformations

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 213–216

As another example consider transformations between HTML tables and an ASCII representation of them. In this case we may like to translate tables to text to allow easier editing by users not familiar with HTML and then translate back to HTML. Additional richness in the source language HTML should not be lost after the translation to text and back to HTML. Thus, it is often necessary to include the original source program as input to the *put* transformation. The idea being that a source phrase is mapped to the target, modified in some way, and then mapped back to the source. By including the original source phrase as input to the *put* transformation a more accurate or realistic source phrase can be computed from the modified target.

In the late 1980's Yellin [8] showed how attribute grammars can specify bidirectional transformations. His *reverse-inverse form* grammars were such that the *put* transformation could be automatically generated from the *get* transformation, specified as attribute definitions. Yellin's approach was proposed before higher order attribute grammars and other modern attribute grammar features were introduced. Thus his *get* and *put* attributes compute phrases as strings of symbols instead of well-typed syntax trees and assumes that only the terminal symbols of the target language are known. Furthermore, the generated *put* transformation does not take as input the original source phrase.

This raises some interesting questions. What if higher-order attributes [7] could be used? What if the nonterminals and productions of the target language are known and used in the definition of the *get* transformation? Furthermore, what if in computing the *put* transformation some information about the source construct is known? For example, is it beneficial to know the source language non-terminal type a target phrase needs to translate back to, or what production in the source language should be used to construct the translation back in the source? Also how can the actual source tree be taken into account by the generated *put* transformation to compute more appropriate translations back to the source? More questions can be raised for bidirectional transformations when additional modern attribute grammars are considered. For example, what benefit do remote [1] and reference [3] attributes provide? What about collection [1] attributes, forwarding [6], and generics [4]? With such features, how can the source be effectively used in computing the *put* transformation?

2 Bi-directional Transformations in Attribute Grammars

To get some idea how answers to these questions might play out consider Figure 2. On the left is the concrete syntax tree for the expression $-2 + 3 * 4$ and on the right is the corresponding abstract syntax tree, generated by the *get* transformation. This transformation is implemented by a collection of higher-order attributes [7] that decorate the source (concrete) nonterminals. These trees correspond to the grammars in Figure 3; nonterminals and productions in the concrete grammar are sub-scripted by "c" to distinguish them from their counterparts in the abstract grammar. Interior nodes of the trees are labelled by the productions that define them. The dashed edges from the abstract to the con-

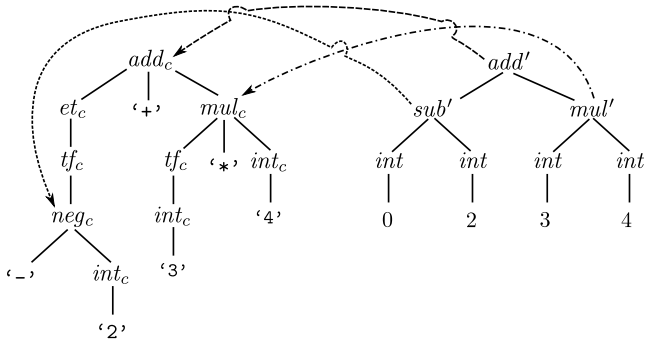


Fig. 2. Concrete and abstract syntax trees, with links back to the source, for $-2+3*4$.

add_c : $E_c ::= E_c\ `+'\ T_c$	tf_c : $T_c ::= F_c$	add : $E ::= E\ E$
sub_c : $E_c ::= E_c\ `-'\ T_c$	int_c : $F_c ::= IntLit_t$	sub : $E ::= E\ E$
et_c : $E_c ::= T_c$	nst_c : $F_c ::= `('\ E_c\ `)'$	mul : $E ::= E\ E$
mul_c : $T_c ::= T_c\ `*'\ F_c$	neg_c : $F_c ::= `-'\ F_c$	int : $E ::= IntLit_t$

Fig. 3. Concrete grammar (first two columns) and abstract (third column) grammar.

crete tree are used to provide access to the original concrete (source) tree to be used by the abstract (target) tree to compute the *put* transformation. These can be realized in attribute grammars as reference [3] or remote [1] attributes. These are effectively pointers to remote nodes in the tree, or in this case, another tree.

We need some mechanism to ensure that the these links are not lost when subtrees are moved or otherwise modified by transformations or optimizations that are performed on the target tree. Such transformations are represented by the downward arrow in Figure 1. One way to accomplish this is to use forwarding [6] and create what amounts to new "wrapper" productions for each original abstract production; this new production takes an extra argument - the reference attribute pointing back to the original source tree. These are "primed" in Figure 2 to distinguish them from the original abstract productions. Attributes defining the *get* transformation can be automatically modified to use these new productions. To compute the *put* transformation of the abstract tree created by the sub' production, we examine the link to the source and determine if this sub' tree was created by neg_c or sub_c.

Consider the rather contrived transformation that switches the order of the two expressions under *add* - a semantically safe transformation given the commutativity of addition. Processes that create the modified tree would use the original *add* production since there would not be an appropriate link back to any source tree that created it. Thus, the new tree node created by the original *add* production would not have a link back to the source, but, for example, its second child (rooted at *sub*) would maintain its link back to the source.

The point here is that modern attribute grammars contain features that can quite naturally be used to specify bidirectional transformations.

We would like to automatically generate the *put* transformation from the specification of the *get* transformation. This has the disadvantage of restricting the specification of the *get* transformation to fall into the class for which a *put* transformation can be generated, but it has the advantage over hand-written *put* transformations in that we can generate *put* transformations that are more complex and sophisticated than one would want to write by hand. For example, in the expression example multiple concrete constructs (E_c, T_c, and F_c) all map to the same abstract construct (E). If we create a *put* attribute on E for each target type, say put_{E_c}, put_{T_c}, and put_{F_c} we can generate a more realistic mapping back to the target in which we do not need to unnecessarily wrap all expressions in parenthesis (the nst_c production) but can do so only when it is required.

3 Conclusion

Yellin's work [8] scratched the surface of what is possible in implementing bidirectional transformations in attribute grammars. With the development of modern attribute grammar features new opportunities are opening for easily implementing expressive transformations between source and target languages. Silver [5] is an extensible attribute grammar system supporting these modern features in which we intend to develop new notations for specifying bidirectional transformations. There are also additional areas of application, specifically in model driven engineering in which transformations between models and programs are common. New mechanisms for implementing bidirectional transformations and new opportunities for their application provide what we believe is a promising vein of research that we are beginning to explore.

References

1. J. T. Boyland. Remote attribute grammars. *J. ACM*, 52(4):627–687, 2005.
2. K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *ICMT '09: Proc. of the 2nd Intl. Conf. on Theory and Practice of Model Transformations*, volume 5563 of *LNCS*, pages 260–283. Springer-Verlag, 2009.
3. G. Hedin. Reference attribute grammars. *Informatica*, 24(3):301–317, 2000.
4. J. Saraiva and D. Swierstra. Generic Attribute Grammars. In *2nd Workshop on Attribute Grammars and their Applications*, pages 185–204, 1999.
5. E. Van Wyk, D. Bodin, J. Gao, and L. Krishnan. Silver: an extensible attribute grammar system. *Science of Computer Programming*, 75(1–2):39–54, January 2010.
6. E. Van Wyk, O. de Moor, K. Backhouse, and P. Kwiatkowski. Forwarding in attribute grammars for modular language design. In *Proc. Intl. Conf. on Compiler Construction*, volume 2304 of *LNCS*, pages 128–142. Springer-Verlag, 2002.
7. H. Vogt, D. Swierstra, and M. Kuiper. Higher order attribute grammars. In *ACM SIGPLAN '89 Conf. on Programming Language Design and Implementation (PLDI)*, pages 131–145. ACM, July 1989.
8. D. M. Yellin. *Attribute Grammar Inversion and Source-to-source Translation*. Number 302 in LNCS. Springer-Verlag, 1988.

Computação Distribuída e de Larga Escala

Curiata: Uma arquitectura P2P auto-organizável para uma localização flexível e eficiente de recursos

João Alverinho, João Leitão, João Paiva, and Luis Rodrigues *

{jalveirinho,jleitao,jgpaiva,ler}@gsd.inesc-id.pt, INESC-ID/IST

Resumo As arquitecturas *entre pares* têm vindo a emergir como uma solução viável para suportar serviços de localização de recursos em sistemas distribuídos de larga escala. A maioria das soluções baseia-se em redes estruturadas (*DHT*s) ou não-estruturadas. As DHTs são mais eficientes para procuras exactas, enquanto que as soluções não-estruturadas apesar de menos eficientes são mais flexíveis. Neste artigo propomos uma nova solução auto-organizável que combina as abordagens estruturada e não-estruturada. Resultados experimentais extraídos através de simulação mostram que a nossa solução consegue oferecer uma boa precisão nas respostas às interrogações, com reduzido custo de mensagens e baixa latência.

Abstract Peer-to-Peer arquitectures have emerged as a viable solution to support resource location services in large-scale distributed systems. Most solutions are based on either structured (*DHT*s) or unstructured overlay networks. DHTs excel on *exact-match* queries, whilst unstructured solutions despite being less efficient are more flexible. In this paper we propose a novel self-organizing solution that combines both structured and unstructured approaches. Experimental results through simulation show that our solution is able to offer good precision in query responses, while keeping a low message cost as well as a low latency.

1 Introdução

Desde o aparecimento do Napster[4] em 1999, os sistemas entre-pares (P2P) têm sido alvo de desenvolvimento e investigação, tanto na academia como na indústria. As aplicações deste tipo de tecnologia incluem partilha de ficheiros[3,1], distribuição de conteúdos[2], partilha de processamento[11], voz sobre IP[5], entre outras. Um serviço fundamental em qualquer sistema P2P, é a localização de recursos. Dado que os sistemas P2P tipicamente almejam suportar um número extremamente elevado de participantes em que cada um destes pode partilhar múltiplos recursos (tempo de CPU, espaço em disco, ficheiros,etc), o espaço de procura pode ser enorme. Soluções centralizadas têm-se provado inviáveis devido a limitações na sua capacidade de escala e fiabilidade. Por outro lado, uma

* Este trabalho foi parcialmente suportado pelo financiamento pluri-anual do INESC-ID através do programa PIDDAC e pelos projectos "Redico" (PTDC/EIA/71752/2006) e "HPCI" (PTDC/EIA-EIA/102212/2008).

procura exaustiva em todos os participantes também não é viável. Assim sendo, o desenho e concretização de serviços distribuídos de localização de recursos é de enorme relevância.

Existem dois tipos principais de sistemas entre-pares: estruturados e não estruturados. Tipicamente, os sistemas P2P estruturados concretizam uma tabela de dispersão distribuída (DHT). Estes sistemas suportam procuras exactas de forma muito eficiente. Contudo, os sistemas estruturados fornecem suporte reduzido à execução de interrogações complexas e/ou inexactas. Adicionalmente, os sistemas estruturados podem ter uma manutenção dispendiosa em ambientes altamente dinâmicos. Uma alternativa passa por usar sistemas não-estruturados, que têm custos de manutenção reduzidos e suportam a execução de qualquer tipo de interrogações sem custos adicionais. No entanto, estes sistemas não conseguem resolver interrogações de forma eficiente. De facto, nos sistemas não-estruturados a localização de recursos é tipicamente realizada através de soluções de procura não guiada (cega), que não produzem resultados satisfatórios no caso geral. Estas abordagens usam mecanismos baseados em inundação (do Inglês, *flooding*), uma solução extremamente dispendiosa e ineficiente; ou em encaminhamento aleatório de uma única mensagem (conhecido como *Random Walk*), uma estratégia com elevada latência e que pode exibir uma fraca Recolha[1] (do Inglês, *Recall*) nos resultados da procura de recursos que não sejam extremamente comuns.

Este artigo apresenta o *Curiata*, um sistema de localização de recursos escalável e eficiente que combina os benefícios das abordagens estruturadas e não-estruturadas. A nossa solução permite flexibilidade nas interrogações, como na maioria das abordagens não-estruturadas, mantendo a rapidez e eficiência providenciada pelas soluções estruturadas (baseadas em DHT's). O modo de operação do *Curiata* inspira-se na organização das sociedades humanas. Durante as duas primeiras décadas da república romana, a população organizava-se em unidades chamadas *curia* de natureza étnica; as *curia* reuniam-se numa assembleia, a *comitia curiata*, para fins legislativos, eleitorais e judiciais, onde os cônsules tinham uma papel especial. De modo análogo, na nossa solução, os participantes organizam-se autonomamente numa rede sobreposta não estruturada onde os nós com recursos semelhantes estabelecem relações de vizinhança através de um processo de baixo custo executado em segundo plano (a *curia*). Para além disto, os nós de cada *curia* elegem representantes para se juntarem a uma rede sobreposta estruturada (a *curiata*). Os membros da rede estruturada são utilizados como ponto de contacto para outros nós com conteúdos semelhantes. Assim sendo, a camada estruturada é utilizada para encaminhar eficientemente as interrogações para regiões da camada não-estruturada que contenham nós que partilhem o tipo de recursos que são procurados. Após ser encaminhada na rede estruturada, a interrogação é propagada pelos membros da *curia* utilizando técnicas de redes não estruturadas como inundação limitada ou encaminhamento aleatório. Além de fornecer uma infra-estrutura que permite a execução eficiente e flexível de interrogações, o *Curiata* também pretende atingir uma *recolha* elevada, com um reduzido custo de mensagens, independentemente da raridade dos recursos.

[1] Esta métrica é defenida em detalhe na secção 4.

Este artigo está organizado da seguinte forma. A Secção 2 fornece uma panorâmica geral dos trabalho relacionado. A Secção 3 descreve o nosso sistema em maior pormenor. Na Secção 4 apresenta-se os resultados da avaliação, enquanto que na Secção 5 registam-se alguns comentários finais que concluem o artigo.

2 Trabalho Relacionado

Os algoritmos de localização de recursos para sistemas P2P têm sido estudados intensivamente existindo diversas soluções propostas na literatura. A abordagem mais simples consiste em adoptar um esquema centralizado, onde um único nó é responsável por manter informação acerca da localização de todos os recursos disponíveis no sistema[4,15]. Dado que um índice central mantém conhecimento global dos recursos disponíveis em todo o sistema, este pode facilmente processar interrogações complexas. Contudo, o custo de processamento imposto a um único nó, para manter informação actualizada sobre todos os recursos do sistema e para processar todas as interrogações geradas por cada participante, pode ser demasiado elevado num ambiente dinâmico de larga escala.

Os sistemas P2P estruturados, que concretizam tabelas de dispersão distribuídas, suportam procuras exactas com um custo em número de mensagens logarítmico com o tamanho do sistema[16,19,18]. No entanto, as DHTs fornecem pouco suporte para interrogações inexactas, dado que decompor uma interrogação complexa em várias interrogações exactas não é trivial e pode até ser impossível. A maioria das soluções existentes (por exemplo [6] e [17]) apesar de permitirem pesquisas mais complexas, apresentam ainda assim uma flexibilidade reduzida ou custos de sinalização e comunicação maiores que os do *Curiata*

Dadas as limitações das redes sobrepostas estruturadas no suporte a interrogações inexactas, torna-se atractivo utilizar redes não-estruturadas, dada a sua maior flexibilidade e menor custo de manutenção. O recurso à inundação com raio limitado é a técnica mais imediata para realizar a localização de recursos sobre redes não-estruturadas[20]. Contudo, esta técnica é muito dispendiosa (devido à duplicação de mensagens) e pode ser ineficiente na localização de recursos raros. As interrogações podem também ser disseminadas recorrendo a percursos aleatórios[14,9], ou encaminhamento informado[8], técnicas menos dispendiosas mas com maior latência e menor recolha. A eficiência das interrogações pode ser melhorada utilizando técnicas como o enviesamento da rede sobreposta para que esta se aproxime de uma rede pequeno-mundo (do inglês, *small-world*), replicando todos os índices de recursos na vizinhança directa de cada nó, e encaminhando as interrogações para nós com maior grau. O GIA[7] é um exemplo conhecido de um sistema que combina estas técnicas. Estas soluções, para além de obrigarem os nós a manter estado adicional, degeneram em configurações onde as interrogações são apenas processadas por uma pequena fracção dos participantes. Adicionalmente, estas soluções não são desenhadas para lidar convenientemente com interrogações que visem recursos raros.

Alguns sistemas propõem a utilização de super-nós[22], onde os participantes se organizam numa hierarquia com dois níveis. Os nós no nível superior mantêm índices consolidados dos recursos partilhados pelos participantes do nível inferior que se ligam a si. Nestes sistemas os super-nós processam a maioria das interrogações e os custos de manutenção dos índices podem facilmente tornar-se

proibitivos num ambiente dinâmico. Apesar de também usarmos uma topologia hierárquica com dois níveis, no *Curiata*, todos os participantes contribuem activamente para a disseminação e processamento das interrogações. Adicionalmente, os participantes não necessitam de manter índices relativos aos seus vizinhos, sendo que, apenas informação genérica sobre as categorias dos recursos dos vizinhos é necessária de forma a enviesar a topologia.

3 *Curiata*

A arquitectura do *Curiata* combina uma camada não-estruturada (a *curia*) com uma camada estruturada (a *comitia curiata*). A nossa solução apresenta cinco componentes principais: i) O *índice de recursos*, que descreve os recursos disponíveis localmente no participante. ii) A camada de *rede não-estruturada enviesada*, que utiliza um protocolo distribuído e auto-organizável para garantir que os participantes estabelecem relações de vizinhança com outros participantes cujos recursos sejam similares. iii) A camada da *rede estruturada*, que é activada somente quando um participante é eleito como cônsul. iv) O *módulo de eleição de cônsules*, que utiliza um protocolo colaborativo para seleccionar os participantes que fazem parte da rede estruturada. v) O *módulo de encaminhamento das procuras*.

Índice de Recursos No *Curiata*, assumimos que os recursos podem ser classificados num conjunto de categorias. O índice de recursos mantém um registo local de todas as categorias dos recursos do participante assim como o número de recursos disponível para cada uma dessas categorias. Esta informação é utilizada para identificar participantes que possuem recursos similares. O esquema de classificação utilizado é ortogonal ao nosso sistema. Por exemplo, uma biblioteca distribuída de artigos sobre informática poderia utilizar o *ACMComputing Classification System* para classificar o conteúdo. Um repositório de música poderia extrair as categorias necessárias para classificar o conteúdo das etiquetas mais utilizadas em aplicações populares como o "Last.fm" (`http://last.fm`).

Adicionalmente, o índice de recursos mantém também, para cada categoria c dos recursos de um participante, a fracção de recursos desse participante que se enquadram nessa categoria. Este valor é denominado como $frac_c$. As categorias são ordenadas de acordo com os valores de *frac*. As primeiras t categorias são utilizadas para definir as relações de vizinhança estabelecidas pelo participante na rede não-estruturada.

Camada de Rede Não-Estruturada O propósito desta camada é organizar todos os participantes que têm recursos disponíveis numa rede sobreposta não-estruturada enviesada. Mais especificamente, propomos que cada participante execute um algoritmo distribuído auto-organizável, para adaptar a topologia da rede sobreposta de acordo com os recursos disponíveis em cada nó. O nosso algoritmo consiste numa versão especializada do X-BOT [12], adaptada para ir de encontro a alguns dos requisitos da nossa arquitectura. O X-BOT é um protocolo distribuído que enviesa a topologia de uma rede não-estruturada simétrica (com as características descritas em [13]) dada uma função de proximidade que forneça uma medida de "distância" entre dois nós no sistema. No caso particular do *Curiata*, a função de proximidade reflecte a similaridade entre os recursos que

dois nós disponibilizam. O objectivo desta estratégia é o de conseguir processar interrogações eficientemente, limitando a procura à área da rede onde existe maior probabilidade de existirem os recursos.

Mais precisamente, a camada da *curia* opera do seguinte modo. Cada nó mantém dois conjuntos de vizinhos, designados por *vista activa* e *vista passiva*. O conjunto de vizinhos da vista activa define a rede que é utilizada para propagar as interrogações. Assim, o tamanho da vista activa d define o grau do nó na *curia*. Os vizinhos da vista passiva são utilizados para explorar a rede e encontrar outros participantes com recursos similares. A vista passiva é actualizada periodicamente por um processo de actualização aleatória das vistas de baixo custo [13]. A vista activa é actualizada através do processo de coordenação introduzido pelo X-BOT utilizando a estratégia que se descreve a seguir.

A *curia* divide a vista activa em t segmentos; cada um destes segmentos dedica-se a uma das primeiras t categorias. O segmento para a categoria c tem uma dimensão $segmento_c$ no intervalo $\lceil smin, d \cdot \text{frac}_c \rceil$. Onde $smin$ é o tamanho mínimo de um segmento, e é definido como $\frac{d}{2t}$. Por exemplo considere-se um sistema onde a *curia* está configurada para seleccionar vizinhos de acordo com as primeiras 5 categorias ($t = 5$). Considere-se um participante p tal que as primeiras 5 categorias $(c1, \ldots, c5)$ têm as seguintes fracções associadas: $(0.5, 0.2, 0.1, 0.1, 0.1)$. Se o participante p apresentar um grau $d = 20$, a sua vista activa seria dividida da seguinte maneira: $(10, 4, 2, 2, 2)$. Considerando este particionamento da vista activa, um nó utiliza o seguinte algoritmo para enviesar os seus vizinhos:

1. O primeiro objectivo do participante quando se junta à rede não estruturada é preencher a sua vista activa, independentemente da similaridade dos seus potenciais vizinhos. Assim sendo, enquanto o número de vizinhos for menor que d, o nó preenche a sua vista activa sem levar em consideração os segmentos definidos para cada categoria.
2. Após ter a vista activa cheia, a próxima prioridade para o participante é ter vizinhos que pertençam às suas t categorias.
3. De seguida, o participante tenta substituir os seus vizinhos por novos vizinhos de modo a que cada segmento da vista activa seja preenchida por participantes com recursos que correspondam à categoria desse segmento.
4. Finalmente, assim que este último critério seja atingido, o participante deixa de executar o algoritmo de auto-organização e mantém os seus vizinhos inalterados enquanto estes não abandonarem a rede.

A selecção do protocolo X-BOT é justificada pela capacidade que este protocolo exibe de enviesar a topologia sem no entanto permitir que a rede apresente um coeficiente de aglomeração excessivo, nomeadamente este protocolo opera de forma a proteger a conectividade global da rede não estruturada (a rede não se particiona em "sub-redes" desconexas entre si). Adicionalmente a operação do protocolo X-BOT garante que o número de vizinhos dos nós se mantém constante.

Camada Estruturada O objectivo da camada estruturada consiste em encaminhar as procuras para zonas da *curia* onde exista maior probabilidade de se

encontrarem os recursos desejados. Para tal, uma fracção dos nós que pertencem à rede não-estruturada também se juntam a uma DHT (como o Chord [19] ou o Pastry [18]). Estes nós são eleitos de forma a representarem uma categoria associada aos recursos dos nós numa dada região da rede não-estruturada, e são designados por *cônsules regionais*. A camada estruturada (a DHT), opera como uma assembleia de representantes de cada uma das diferentes regiões no espaço não-estruturado.

Eleição de Cônsules Em cada região de raio r na rede não-estruturada, se existe uma categoria c que é a principal categoria de um nó (ou seja, a categoria com a maior fracção na vista activa desse participante) nessa região, existe um participante que representa c na DHT. Esse participante designa-se por *cônsul regional* para a categoria c, ou simplesmente c-cônsul. Um c-cônsul junta-se à DHT com um identificador construído através da concatenação dos bits mais significativos de *hash(c)* com os bits menos significativos de *hash(node_id)*. Isto garante que múltiplos contactos para a mesma categoria, em diferentes regiões, têm identificadores diferentes mas, posicionam-se numa região consecutiva no espaço de endereços da DHT.

Um c-cônsul usa a rede não-estruturada para periodicamente enviar um *sinal* para os nós na sua vizinhança de raio r. Nós que recebem este sinal abstêm-se de competir para se tornarem c-cônsules. Se um nó que possui a categoria c como a sua principal categoria não receber nenhum sinal durante um determinado intervalo de tempo, este decide competir com outros potenciais candidatos para se tornar um contacto regional para c.

Para tal, esse nó envia para os nós na sua vizinhança de raio r um *sinal* de *promoção* a cônsul. Passado um intervalo de tempo t pré-definido, caso o nó não tenha recebido nenhum *sinal* de *promoção* proveniente de outro nó, ele considera que a sua *promoção* teve sucesso e tenta juntar-se à camada estruturada. Quando múltiplos nós competem, um protocolo de eleição (tipo *bully*) é utilizado para seleccionar que nó se torna c-cônsul. Por exemplo, dando prioridade ao nó que possua mais recursos da categoria a que se candidata.

Um nó que é eleito para ser um c-cônsul, pode utilizar um outro cônsul de uma categoria c' como ponto de contacto para se juntar à DHT ou, se não conhecer nenhum, realiza um percurso aleatório na rede não-estruturada para encontrar um nó que tenha o contacto de um cônsul.

Procuras Seguidamente descreve-se a forma como são executadas procuras no nosso sistema. A nossa arquitectura não restringe o formato nem a linguagem utilizada nas interrogações. Existe apenas um requisito: a partir da interrogação deve ser possível extrair o conjunto $\mathcal{Q}$ de categorias relevantes aos recursos alvo da interrogação se direcciona. Por exemplo, assuma-se que a interrogação procura por uma música por Aldina Duarte; assim sendo, terá que ser possível extrair categorias como *Música*, *Fado* e *Portugal*.

Uma procura é executada do seguinte modo: i) Primeiro, a interrogação é encaminhada para um membro da DHT; ii) Seguidamente uma cópia da interrogação é encaminhada para cada categoria $c \in \mathcal{Q}$ utilizando a DHT. Cada cópia será recebida por um c-cônsul para essa categoria. De modo a promover balanceamento de carga, para cada categoria c, a interrogação é encaminhada para um

identificador composto por *hash(c)*||{s bits aleatórios}. Isto garante que diferentes interrogações são injectadas na camada não-estruturada através de diferentes representantes dessa categoria; iii) Cada c-cônsul inicia um percurso aleatório de comprimento $\frac{k}{|\mathcal{Q}|}$ na sua vizinhança. Estes percursos são guiados, sendo a interrogação apenas encaminhada para vizinhos que possuam recursos da categoria c; iv) Cada nó visitado pela interrogação executa-a e, caso possua os recursos procurados, adiciona o seu identificador à interrogação; v) Finalmente, quando a interrogação efectua um número máximo de saltos na rede, é retornada à fonte a lista de todos os nós encontrados que satisfazem a interrogação.

O objectivo é processar cada interrogação com um custo de mensagem aproximado k. No protótipo actual, o valor de k é estático. Contudo, k poderia ser ajustado dinamicamente consoante uma estimativa da raridade do recurso a ser procurado (por exemplo, baseado nos resultados retornados por procuras anteriores). Para recursos mais comuns um valor de k mais limitado poderá ser suficiente para a localização dos mesmos, enquanto que para recursos raros pode ser útil aumentar o número de saltos que o percurso aleatório pode efectuar na rede.

O custo total de mensagens de uma interrogação é a soma de $\frac{k}{|\mathcal{Q}|} \times |\mathcal{Q}|$, com o custo de chegar a um membro da DHT a partir da fonte (tipicamente 1 salto), e o número de saltos na DHT necessários para chegar ao c-cônsul respectivo (na ordem de $c \cdot \ln T$ onde T é o número de nós na DHT).

É possível configurar o sistema de modo a promover um valor baixo de T, ajustando os valores de d (número de vizinhos de cada nó) e de r (raio do sinal enviado pelos cônsules). Como trabalho futuro, pretendemos estudar formas de ajustar os valores destes parâmetros em tempo de execução, por exemplo de forma a diminuir o valor de r no caso de existir um pico ao número de interrogações efectuadas, promovendo um aumento do número de cônsules e assim, um melhor balanceamento de carga ao nível dos nós na DHT.

4 Avaliação Experimental

Nesta secção apresentam-se os resultados obtidos com uma concretização do nosso sistema para o simulador *Peersim*[10]. Para as experiências foi utilizada uma rede com 10.000 nós e 11 categorias. A cada nó no sistema é atribuída uma categoria das onze. Os recursos na rede estão também eles associados a uma única categoria e possuem um identificador único. Os recursos de cada categoria c são alocados aleatoriamente em nós nessas categorias, existindo 5 nós distintos na rede que tenham esse recurso. Configuraram-se os nós para terem $d = 20$. A Tabela 1 sumariza o número de nós em cada categoria e o número total de recursos únicos associados a cada categoria. O raio das regiões para eleição de cônsules foi configurado com um valor de 2 e, neste cenário aproximadamente 200 participantes são eleitos como cônsules e juntam-se à camada estruturada(DHT). Consideramos ainda que cada categoria se insere numa das 3 classes de popularidade que se seguem: *Categorias Raras:* categorias com menos de 100 nós (Categorias G a K); *Categorias Intermédias:* categorias com 100 a 1.000 nós (Categorias D a F); *Categorias Comuns:* categorias com mais de 1.000 nós (Categorias A a C).

Nestas experiências foram realizadas 10.000 procuras, com origem em nós aleatórios na rede, que foram definidas de forma a possuírem um único recurso alvo. Mais à frente serão apresentados os resultados para categorias comuns e raras.

Arquitecturas Avaliadas Comparamos o desempenho de quatro arquitecturas diferentes:

Percursos Aleatórios numa Topologia Aleatória (PATA): Corresponde a um sistema que utiliza apenas uma rede não-estruturada aleatória, onde não se aplica um processo de auto-organização. Nesta arquitectura, não existe DHT e os vizinhos não dependem da similaridade dos seus recursos.

Percursos Aleatórios Guiados numa Topologia Aleatória (PAGTA): Corresponde a um sistema que usa uma rede não-estruturada aleatória, onde não existe enviesamento da topologia. Contudo, nesta arquitectura as procuras são guiadas utilizando um mecanismo similar ao utilizado no *Curiata*.

Percursos Aleatórios Guiados numa Topologia Enviesada (PAGTE): Corresponde a um sistema que utiliza uma rede não-estruturada enviesada pelo mesmo processo de auto-organização utilizado no *Curiata*. Nesta arquitectura as procuras também são guiadas. Contudo não existe DHT para encaminhar as interrogações.

Curiata: Concretização completa da arquitectura descrita neste artigo.

Interrogações Nas simulações, cada interrogação é concretizada recorrendo a um único percurso aleatório guiado com comprimento $k = 128$ ou $k = 256$. Este comprimento é medido a partir do nó que produz a interrogação. Assim, todos os saltos na rede são considerados, incluindo os saltos necessários para atingir o cônsul mais perto na DHT para a categoria referente a cada interrogação.

Para simplificar a análise dos resultados, cada interrogação nas simulações procura apenas por um único recurso. Dado que cada recurso está associado a apenas uma categoria, esta categoria é utilizada para guiar a interrogação tanto na arquitectura PAGTA, como na arquitectura PAGTE e também no *Curiata*. Note-se que o facto de os recursos possuírem identificadores únicos é apenas um artefacto de simulação. Como referido anteriormente, o *Curiata* permite a utilização de interrogações arbitrariamente complexas. Assim, a procura por um recurso específico simula uma qualquer interrogação complexa que é satisfeita apenas pelo recurso com esse identificador.

Métricas Nesta avaliação experimental consideramos as seguintes métricas: i) *Taxa de Sucesso:* Percentagem de interrogações que encontram pelo menos

Categoria	Nm. de nós	Recursos Únicos	Categoria	Nm. de nós	Recursos Únicos
A	5000	5	*G*	75	5
B	2500	5	*H*	40	5
C	1250	5	*I*	30	5
D	625	5	*J*	20	4
E	300	5	*K*	10	2
F	150	5			

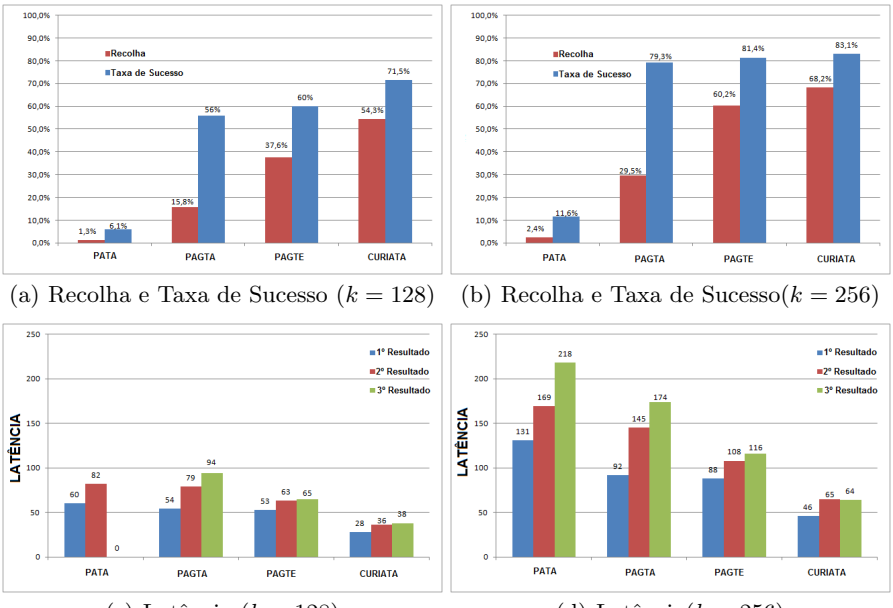
Tabela 1. Distribuição de recursos na rede

uma cópia do recurso procurado; ii) *Recolha:* Percentagem de cópias do recurso procurado que é encontrada pela interrogação (em comparação com o número total de cópias desse recurso existentes no sistema); iii) *Latência:* Número de saltos necessários para encontrar x cópias do recurso procurado. Em particular, estamos interessados nos valores de latência associados a encontrar 1, 2 e 3 cópias do recurso.

Desempenho Global Apresentam-se agora os resultados globais para as interrogações no cenário descrito acima. Neste caso os resultados não são discriminados de acordo com as diferentes classes de popularidade das categorias, dado que o objectivo é fornecer uma visão geral do desempenho do *Curiata*.

Como ilustrado nas figuras Fig 1(a) e Fig 1(b), o *Curiata* possui um desempenho superior às restantes arquitecturas em termos de taxas de sucesso e recolha. Os valores de recolha para a arquitectura PATA quase duplicam quando k passa de 128 para 256 dado que os percursos aleatórios conseguem explorar o dobro dos nós. Note-se no entanto que o desempenho do *Curiata* é muito melhor que as restantes soluções com $k = 128$. Isto é um claro sinal de que o *Curiata* consegue atingir bons resultados mesmo com valores mais conservadores de k.

Naturalmente, num caso limite em que k seja um valor suficientemente grande, todas as arquitecturas (PATA,PAGTA,PAGTE e *Curiata*) conseguiriam encontrar todos os recursos existentes na rede, atingindo taxas de Recolha e Sucesso de 100%. No entanto a eficiência do *Curiata*, atingindo elevadas taxas de sucesso e recolha, para valores baixos de k é um sinal claro do menor custo (em termos de comunicação) da nossa solução.



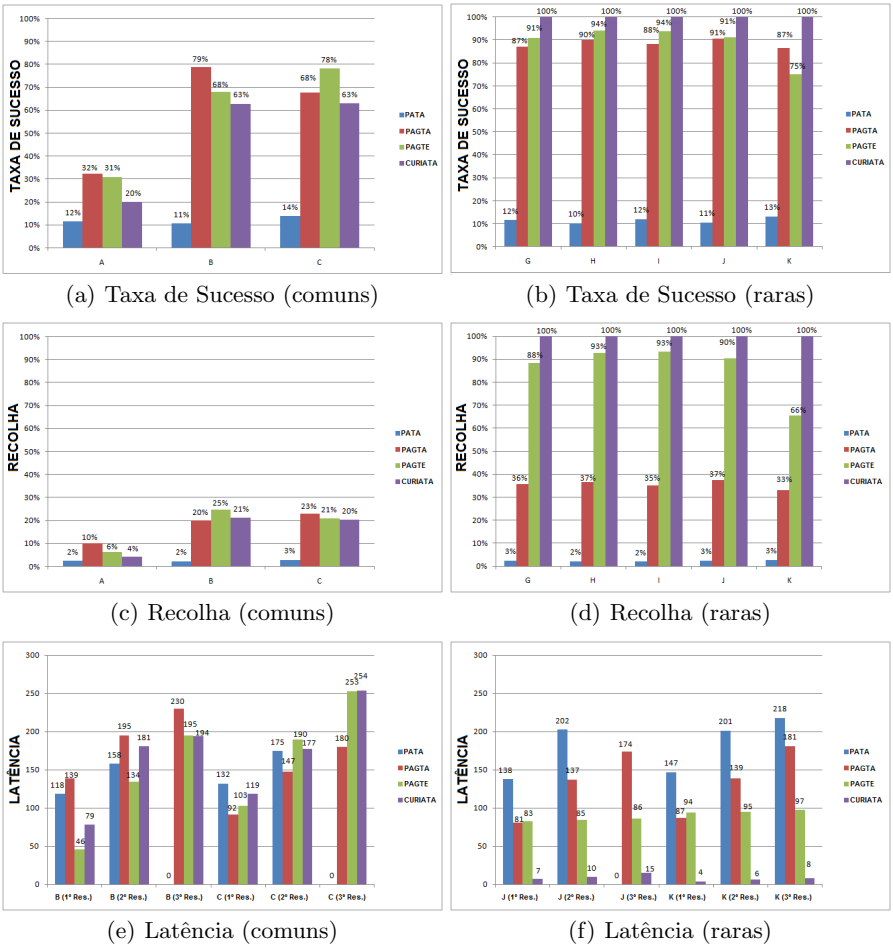
(a) Recolha e Taxa de Sucesso ($k = 128$) (b) Recolha e Taxa de Sucesso($k = 256$)

(c) Latência ($k = 128$) (d) Latência($k = 256$)

Figura 1. Desempenho Global

As figuras Fig 1(c) e Fig 1(d) apresentam os valores de latência. Devido ao encaminhamento na DHT, o *Curiata* apresenta valores de latência significativamente menores. Note-se que a falta de suporte da DHT leva a um cenário onde não existem diferenças significativas no número de saltos necessários para encontrar o primeiro resultado para uma interrogação. Adicionalmente, o enviesamento da topologia da *curia* combinado com a utilização da DHT permite ao *Curiata* apresentar valores de latência mais baixos associados à localização do segundo e terceiro resultados para cada interrogação. Note-se que o valor de latência para o terceiro resultado utilizando PATA é igual a zero dado que esta solução não localiza 3 resultados numa mesma interrogação para k=128. O mesmo não sucede para k=256.

Dado que o rácio de sucesso não é de 100%, quando se altera o valor de k de 128 para 256 o número de saltos na rede aumenta. Isto sucede devido ao au-



(a) Taxa de Sucesso (comuns)

(b) Taxa de Sucesso (raras)

(c) Recolha (comuns)

(d) Recolha (raras)

(e) Latência (comuns)

(f) Latência (raras)

Figura 2. Desempenho das Procuras na Categorias Raras e Comuns

mento do número de interrogações que de facto conseguem localizar pelo menos um resultado. Note-se que estes resultados apenas tomam em consideração o número de saltos para interrogações que tiveram sucesso (isto explica os resultados apresentados na Fig 1(d), onde o número de saltos na rede necessários para o terceiro resultado é menor do que para o segundo resultado quando $k = 256$ utilizando o *Curiata*).

Desempenho por Popularidade da Categoria A Fig 2 apresenta os resultados para interrogações que visam recursos em categorias raras (nomeadamente, as categorias G, H, I, J, e K) e categorias comuns (as categorias A, B, e C). Devido a constrangimentos de espaço apenas exibimos os resultados para $k = 256$, já que este é o melhor cenário para as restantes soluções consideradas.

Os resultados obtidos mostram que o *Curiata* não oferece vantagens significativas para interrogações que visam recursos em categorias comuns. Isto sucede porque a DHT não é necessária quando as categorias são muito populares (dado que os recurso de categorias comuns estão disponíveis em todas as regiões). De facto, a performance do *Curiata* para categorias comuns é até ligeiramente inferior à das arquitecturas PAGTE e PAGTA. Tal sucede dado que se torna desnecessária a utilização da camada estruturada para encaminhar as interrogações para zonas da *curia* correspondentes a essas categorias (dado estas serem muito comuns).

Em contraste, o *Curiata* distingue-se no caso particular de interrogações relativas a recursos pertencentes a categorias raras. Neste caso, o *Curiata* consegue atingir uma recolha e uma taxa de sucesso perfeitos sendo o seu desempenho superior a todas as outras soluções em termos de latência. A Fig 2(f) mostra pormenorizadamente os resultados para a latência apenas para as categorias raras J e K. Os resultados mostram que o *Curiata* oferece um ganho ao nível da latência muito significativo quando comparado com as restantes alternativas. Isto advém do facto de a DHT posicionar as interrogações na região relevante da camada não-estruturada de uma forma bastante precisa. Dado que estas regiões são pequenas, o *Curiata* consegue facilmente visitar todos os nós relevantes e assim localizar eficientemente todas as cópias dos recursos que são visados por cada interrogação.

5 Conclusões

Neste artigo introduzimos o *Curiata*, uma arquitectura para localização de recursos em sistemas P2P de larga escala. O *Curiata* combina técnicas de redes estruturadas e não-estruturadas de modo a oferecer uma elevada recolha e baixa latência para interrogações complexas que visam localizar recursos raros. O *Curiata* não apresenta uma perda de desempenho na resolução de interrogações relativas a recursos raros em categorias comuns quando comparado com outros esquemas. Estes resultados indicam que, quando se procura por um recurso que pertence a múltiplas categorias, são obtidos melhores resultados quando as categorias mais raras são utilizadas de forma preferencial. Como trabalho futuro, planeamos explorar mais aplicações para o *Curiata*. Por exemplo, planeamos integrar a DHT *Cubit*[21] no *Curiata* para desenvolver uma infra-estrutura descentralizada de rastreio e localização de *torrents* que consiga lidar com erros cometidos pelos utilizadores quando descrevem e classificam os conteúdos.

Referências

1. Bittorrent. `http://bittorrent.org/beps/bep_0003.html`.
2. Coral. `http://www.coralcdn.org/`.
3. Gnutella. `http://rfc-gnutella.sourceforge.net/`.
4. Napster. `http://www.napster.com`.
5. *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol*, 2006.
6. A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *In Proc. of the 2nd P2P'02*, page 33, Washington, DC, USA, 2002. IEEE Comp. Society.
7. Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proc. of the 2003 conference*, pages 407–418, New York, NY, USA, 2003. ACM.
8. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of the 22nd ICDCS'02*, pages 23–32, 2002.
9. C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: algorithms and evaluation. *Perform. Eval.*, 63(3):241–263, 2006.
10. M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. The Peersim simulator. `http://peersim.sf.net`.
11. E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. Seti@home-massively distributed computing for seti. *Computing in Science and Engineering*, 3(1):78–83, Jan/Feb 2001.
12. J. Leitão, J. P. Marques, J. Pereira, and L. Rodrigues. X-bot: A protocol for resilient optimization of unstructured overlays. In *Proc. of the 28th IEEE SRDS'09*, pages 236–245, Niagara Falls, New York, U.S.A., September 2009.
13. J. Leitão, J. Pereira, and L. Rodrigues. Hyparview: a membership protocol for reliable gossip-based broadcast. In *Proc. of the 37th IEEE/IFIP DSN'07*, pages 419–429, Edinburgh, UK, June 2007.
14. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.
15. R. Raman, M. Livny, and M. Solomon. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2):129–138, 1999.
16. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. of the 2001 ACM SIGCOMM Conference*, volume 31, pages 161–172, New York, NY, USA, October 2001. ACM.
17. P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *(Unpublished Manuscript)*, pages 21–40, 2002.
18. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Int. Conf. on Distributed Systems Platforms*, pages 329–350, November 2001.
19. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01*, pages 149–160, New York, NY, USA, 2001. ACM.
20. D. Tsoumakos and N. Roussopoulos. Analysis and comparison of p2p search methods. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 25, New York, NY, USA, 2006. ACM.
21. B. Wong, A. Slivkins, and E. G. Sirer. Approximate matching for peer-to-peer overlays with cubit. Technical report, Computing and Information Science Technical Report, Cornell University, Dec. 2008.
22. Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. *Data Engineering, International Conference on*, 0:49, 2003.

Evaluating Data Freshness in Large Scale Replicated Databases*

Miguel Araújo and José Pereira

Universidade do Minho

Abstract. There is nowadays an increasing need for database replication, as the construction of high performance, highly available, and large-scale applications depends on it to maintain data synchronized across multiple servers. A particularly popular approach, used for instance by Facebook, is the MySQL open source database management system and its built-in asynchronous replication mechanism. The limitations imposed by MySQL on replication topologies mean that data has to go through a number of hops or each server has to handle a large number of slaves. This is particularly worrisome when updates are accepted by multiple replicas and in large systems.
It is however difficult to accurately evaluate the impact of replication in data freshness, since one has to compare observations at multiple servers while running a realistic workload and without disturbing the system under test. In this paper we address this problem by introducing a tool that can accurately measure replication delays for any workload and then apply it to the industry standard TPC-C benchmark. This allows us to draw interesting conclusions about the scalability properties of MySQL replication.

Keywords: Databases, Replication, MySQL, Data Freshness

1 Introduction

With the rapid growth of the Internet, availability has recently became critical due to large amounts of data being captured and used each day with the emerging online services. Large companies such as Google, eBay, or Amazon handle exabytes of data per year. Facebook claims to be one of the largest MySQL installations running thousands of servers handling millions of queries, complemented by its own Cassandra data store for some very specific queries. These Internet-based services have become a standard in our information society, supporting a wide range of economic, social, and public activities. And in this globalized era, since large organizations are present in different places all over the world, information must be always online and available. The loss of information or its unavailability can lead to serious economic damages. So, high-availability, performance, and reliability are all critical requirements in such systems.

* Partially funded by project ReD – Resilient Database Clusters (PDTC / EIA-EIA / 109044 / 2008).

Both of these challenges are commonly addressed by means of the same technique, namely data replication. Application components must be spread over a wide area network, providing solutions that enable high availability through network shared contents. Since databases are more and more deployed on clusters and over wide area networks, replication is a key component. Replicating data improves fault-tolerance since the failure of a site does not make a data item inaccessible. Available sites can take over the work of failed ones. And also improves performance since data access can be localized over the database network, i.e. transaction load is distributed across the replicas, achieving load balancing, and in the other hand it can be used to provide more computational resources, or allow data to be read from closer sites reducing the response time and increasing the throughput of the system.

There are however several different replication protocols, differing first and foremost whether propagation takes place within transaction boundaries [3]: Lazy schemes use separate transactions for execution and propagation, in contrast to eager schemes that distribute updates to replicas in the context of the original updating transaction. Thus, the eager method makes it easy to guarantee transaction properties, such as serializability but, since such transactions are distributed and relatively long-lived, the approach does not scale well [2]. On the other hand, lazy replication reduces response times as transactions can be executed and committed locally and only then propagated to other sites [4]. In detail, being replicated asynchronously, data is first written on the master server and then is propagated to slaves, and so, specially in the case of hundreds of servers, slaves will take some time to obtain the most recent data. Lazy propagation thus opens up the possibility of having stale data in replicas and makes data freshness a key issue for correctness and performance.

Most database management systems implement asynchronous master-slave replication. The systems provide mechanisms for master-slave replication that allows configuring one or more servers as slaves of another server, or even to behave as master for local updates. MySQL in particular allows almost any configuration of master and slaves, as long as each server has at most one master. As described in Section 2, this usually leads to a variety of hierarchical replication topologies, but includes also a ring which allows updates to be performed at any replica, as long as conflicts are avoided.

It is thus interesting to assess the impact of replication topology in MySQL, towards maximizing scalability and data freshness. This is not however easy to accomplish. First, it requires comparing samples obtained at different replicas and thus on different time referentials, or, when using a centralized probe, network round-trip has to be accounted for. Second, the number of samples that can be obtained has to be small in order not to introduce a probing overhead. Finally, the evaluation should be performed while the system is running a realistic workload, which makes it harder to assess the point-in-time at each replica with a simple operation. In this paper we address these challenges by making the following contributions:

– We describe a tool that obtains a small number of samples of log sizes using a centralized probe at different points in time. It then selects particularly interesting periods of time and computes a freshness value with the distance between lines fitted to these points.
– We apply the tool to two representative MySQL configurations with a varying number of replicas and increasingly large workloads using the industry standard TPC-C on-line transaction processing benchmark [1]. This allows us to derive conclusions on the scalability of MySQL replication.

The rest of the paper is structured as follows: Section 2 describes the MySQL replication architecture in detail. In Section 3, the method and tool used to achieve this goal is presented. Section 4 presents the results obtained and Section 5 concludes the paper.

2 Background

MySQL, as most database management systems do, implements asynchronous master-slave replication. It allows configuring each server as slave of any other server while simultaneously behaving as master for local updates. The configuration of replication allows an arrangement of masters and slaves in different topologies and it is possible to replicate the entire server, replicate only certain databases or to choose what tables to replicate.

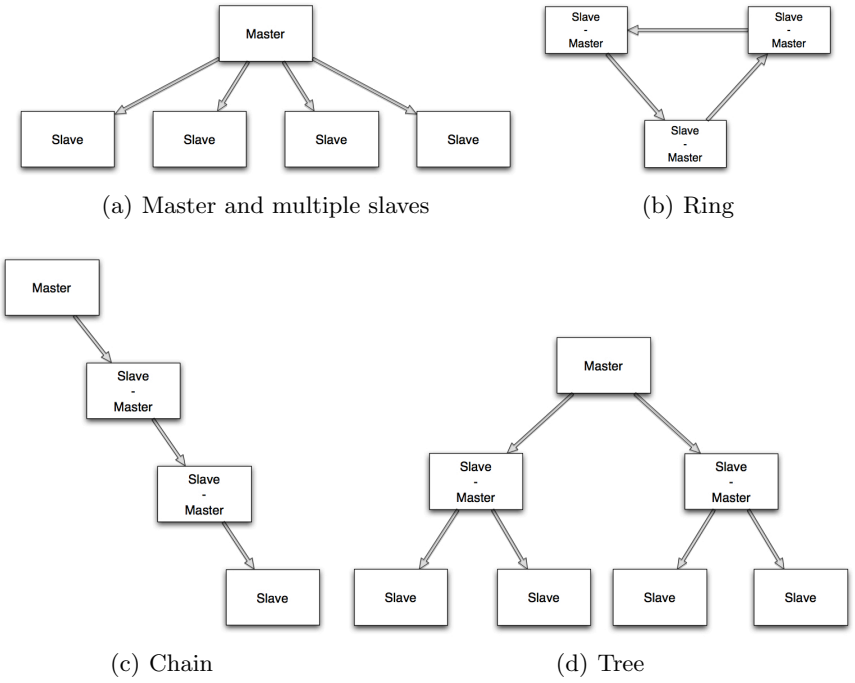
2.1 Replication Mechanism

The replication mechanism of MySQL, works at a high level in a simple three-part process:

1. The master records changes to its data in its binary log (these records are called binary log events).
2. The slave copies the master's binary log events to its own log (relay log).
3. The slave replays the events in the relay log, applying the changes to its own data.

Briefly, after writing the events to the binary log, the master tells the storage engine to commit the transactions. The next step is for the slave to start an I/O thread to start the dump. This process reads events from the master's binary log. If there are events on the master, the thread writes them on the relay log. Finally, a thread in the slave called SQL thread reads and replay events from the relay log, thus updates slave's data to match the master's data. To notice that the relay log usually stays in the operating system's cache, having very low overhead.

This replication architecture decouples the processes of fetching and replaying events on the slave, which allows them to be asynchronous. That is, the I/O thread can work independently of the SQL thread. It also places constraints on the replication process, the most important of which is that replication is



(a) Master and multiple slaves (b) Ring

(c) Chain (d) Tree

Fig. 1. Sample MySQL replication topologies.

serialized on the slave. This means updates that might have run in parallel (in different threads) on the master cannot be parallelized on the slave, which is a performance bottleneck for many workloads.

2.2 Replication Topologies

The simplest topology besides **Master-Slave** is **Master and Multiple Slaves** (Figure 1(a)). In this topology, slaves do not interact with each other at all, they all connect only to the master. This is a configuration useful for a system that has few writes and many reads. However, this configuration is scalable to the limit that the slaves put too much load on the master or network bandwidth from the master to the slaves becoming a problem.

Other possible configuration is **Master-Master in Active-Active Mode**. This topology involves two servers, each configured as both a master and slave of the other. The main bottleneck in this configuration resides on how to handle conflicting changes.

A variation on master-master replication that avoids the problems of the previous is the **Master-Master in Active-Passive** mode replication. The main difference is that one of the servers is a read-only "passive" server. This configuration permits swapping the active and passive server roles back and forth very

easily, because the servers configurations are symmetrical. This makes failover and failback easy.

The related topology of the previous ones is **Master-Master with Slaves**. The advantage of this configuration is extra redundancy. In a geographically distributed replication topology, it removes the single point of failure at each site.

One of the most common configuration in database replication, is the **Ring** topology (Figure 1(b)). A ring has three or more masters. Each server is a slave of the server before it in the ring, and a master of the server after it. This topology is also called circular replication. Rings do not have some of the key benefits of a master-master setup, such as symmetrical configuration and easy failover. They also depend completely on every node in the ring being available, which greatly increases the probability of the entire system failing. And if you remove one of the nodes from the ring, any replication events that originated at that node can go into an infinite loop. They will cycle forever through the topology, because the only server that will filter out an event based on its server ID is the server that created it. In general, rings are brittle and best avoided. Some of the risks of ring replication can be decreased by adding slaves to provide redundancy at each site. This merely protects against the risk of a server failing, though.

Another possibility, regarding some certain situations where having many machines replicating from a single server requires too much work for the master, or the replication is to spread across a large geographic area that chaining the closest ones together gives better replication speed, is the **Daisy Chain** (Figure 1(c)). In this configuration each server is set to be a slave server to one machine as as master to another in a chain. Again, like the ring topology the risk of losing a server can the decreased by adding slaves to provide redundancy at each site.

The other most common configuration is the **Tree or Pyramid** topology (Figure 1(d)). This is very useful in the case of replicating a master to a very large number of slaves. The advantage of this design is that it eases the load on the master, just as the distribution master did in the previous section. The disadvantage is that any failure in an intermediate level will affect multiple servers, which would not happen if the slaves were each attached to the master directly. Also, the more intermediate levels you have, the harder and more complicated it is to handle failures.

2.3 Data Freshness

Data replication must ensure ACID properties and copy consistency must be preserved through global isolation [6]. To ensure global isolation a transaction that modifies data must update all its copies before any other transaction can access the data. Property known as 1-copy serializability. This property can be ensured with synchronous replication, in which a transaction updates all replicas, enforcing the mutual consistency of all replicas. However, this replication model increases the transaction latency because extra messages are added to the transaction (distributed commit protocol).

On the other hand, lazy replication updates all the copies in separate transactions, so the latency is reduced in comparison with eager replication. A replica is updated only by one transaction and the remain replicas are updated later on by separate refresh transactions [7].

Although there are concurrency control techniques and consistency criterion which guarantee serializability in lazy replication systems, these techniques do not provide data freshness guarantees. Since transactions may see stale data, they may be serialized in an order different from the one in which they were submitted.

So, asynchronous replication leads to periods of time that copies of the same data diverge. Some of them have already the latest data introduced by the last transaction, and others have not. This divergence leads to the notion of data freshness: The lower the divergence of a copy in comparison with the other copies already updated, the fresher is the copy [5].

MySQL replication is commonly known as being very fast, as it depends strictly on the the speed that the engine copies and replays events, the network, the seize of the binary log, and time between logging and execution of a query [8]. However, there have not been many systematic efforts to precisely characterize the impact on data freshness.

One approach is based on the use of a User Defined Function returning the system time with microsecond precision [8]. Inserting this function's return value on the tables we want to measure and comparing it to the value on the respective slave's table we can obtain the time delay between them. But this measurements can only be achieved on MySQL instances running on the same server due to clock inaccuracies between different machines.

A more practical approach uses a Perl script and the Time::HiRes module to get the system time with seconds and microseconds precision.[1] The first step is to insert that time in a table on the master, including the time for the insertion. After this, the slave is queried to get the same record and immediately after the attainment of it the subtraction between system's date and time got from the slave's table is made, obtaining the replication time. As with the method described above this one lacks of accuracy due to the same clock inaccuracies.

3 Measuring Propagation Delay

3.1 Approach

Our approach is based on using a centralized probe to periodically query each of the replicas, thus discovering what has been the last update applied. By comparing such positions, it should be possible to discover the propagation delay. There are however several challenges that have to be tackled to obtain correct results, as follows.

[1] http://datacharmer.blogspot.com/2006/04/measuring-replication-speed.html

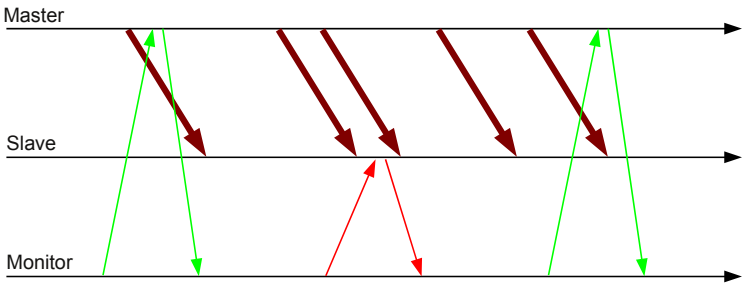


Fig. 2. Impossibility to probe simultaneously master and slaves.

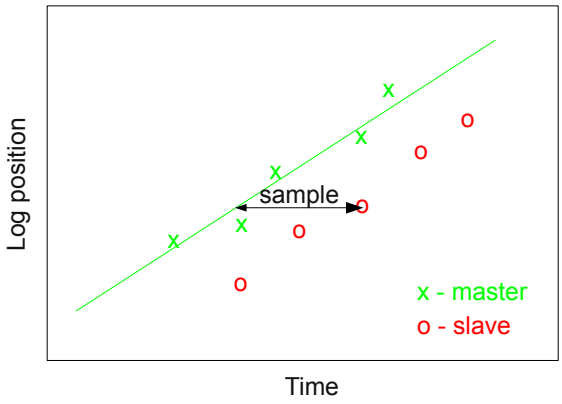


Fig. 3. Log position over the time

Measuring updates. The first challenge is to determine by how much two replicas differ and thus when two replicas have applied exactly the same amount of updates. Instead of trying to compare database content, which would introduce a large overhead, or using a simple database schema and workload that makes it easy, we use the size of the transactional log itself. Although this does not allow us to measure logical divergence, we can determine when two replicas are exactly with the same state.

Non-simultaneous probing. The second challenge is that, by using a single centralized probe one cannot be certain that several replicas are probed at exactly the same time. Actually, as shown in (Figure 2), if the same monitor periodically monitors several replicas it is unlikely that this happens at all. This makes it impossible to compare different samples directly.

Instead, as shown in (Figure 3) we consider time–log position pairs obtained by the monitor and fit a line to them (using the least-squares method). We can then compute the distance of each point obtained from other replicas to this line along the time axis. This measures how much time such replica was stale.

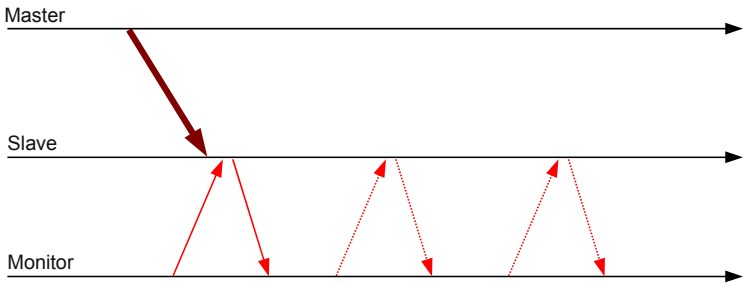


Fig. 4. Sampling twice without updates erroneously biases the estimate.

Eliminating quiet periods. Moreover, as replication traffic tends to be bursty. If one uses repeated samples of a replica that stands still at the same log position, the estimate is progressively biased towards a (falsely) higher propagation delay, as shown in (Figure 4). This was solved by selecting periods where line segments obtained from both replicas have a positive slope, indicating activity.

Dealing with variability. Finally, one has to deal with variability of replication itself and over the network used for probing. This is done by considering a sufficient amount of samples and assuming that each probe happens after half of the observed round-trip. Moreover, a small percentage of the highest round-trips observed is discarded, to remove outliers.

3.2 Implementation

An application to interrogate the master instance and several replicas of the distributed database scheme was developed. This tool stores the results in a file for each instance. To obtain the log position it uses the MySQL API in order to obtain the replication log position. The temporal series of observed log positions are then stored in separate files, one for each node of the distributed database.

Results are then evaluated off-line using the Python programming language and R statistics package. This script filters data as described and then adjusts a line to the values of the log files and compares them. This includes looking for periods of heavy activity and fitting line segments to those periods. With these line segments, the script compares each slave points with the corresponding segment on the master, if the segment does not exist for the selected point, the point is ignored. In the end, average is calculated based on the difference of values between slave points and corresponding segments on the master. A confidence interval can also be computed, using the variance computed from the same data.

4 Experiments

4.1 Workload

In order to assess the distributed database used in the case study, we have chosen the workload model defined by TPC-C benchmark [1], a standard on-line transaction processing (OLTP) benchmark which mimics a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. Specifically, we used the Open-Source Development Labs Database Test Suit 2 (DBT-2), a fair usage implementation of the specification.

Although TPC-C includes a small amount of read-only transactions, it is composed mostly by update intensive transactions. This choice makes the master server be almost entirely dedicated to update transactions even in a small scale experimental setting, mimicking what would happen in a very large scale MySQL setup in which all conflicting updates have to be directed at the master while read-only queries can be load-balanced across all remaining replicas.

Each client is attached to a database server and produces a stream of transaction requests. When a client issues a request it blocks until the server replies, thus modeling a single threaded client process. After receiving a reply, the client is then paused for some amount of time (think-time) before issuing the next transaction request. The TPC-C model scales the database according to the number of clients. An additional warehouse should be configured for each additional ten clients. The initial sizes of tables are also dependent on the number of configured clients.

During a simulation run, clients log the time at which a transaction is submitted, the time at which it terminates, the outcome (either abort or commit) and a transaction identifier. The latency, throughput and abort rate of the server can then be computed for one or multiple users, and for all or just a subclass of the transactions. The results of each DBT-2 run include also CPU utilization, I/O activity, and memory utilization.

4.2 Setting

Two replication schemes were installed and configured. A five machines topology of master and multiple slaves, and a five machine topology in daisy chain.

The hardware used included six HP Intel(R) Core(TM)2 CPU 6400 - 2.13GHz processor machines, each one with one GByte of RAM and SATA disk drive. The operating system used is Linux, kernel 2.6.31-14, from Ubuntu Server, and the database engine used is MySQL 5.1.39. All machines are connected through a LAN, and are named PD01 to PD06. Being PD01 the master instance, PD04 the remote machine in which the interrogation client executes, and the others the slave instances.

The following benchmarks were done using the workload TPC-C with the scale factor (warehouses) of two, number of database connections (clients) one hundred and the duration of twenty minutes.

Replica	PD02	PD03	PD05	PD06
Number of samples	15238	15121	15227	15050
Average delay (μs)	10133	10505	10249	10260
99% confidence interval ($\pm$)	363	373	412	378

Table 1. Results for master and multiple slaves topology with 100 clients.

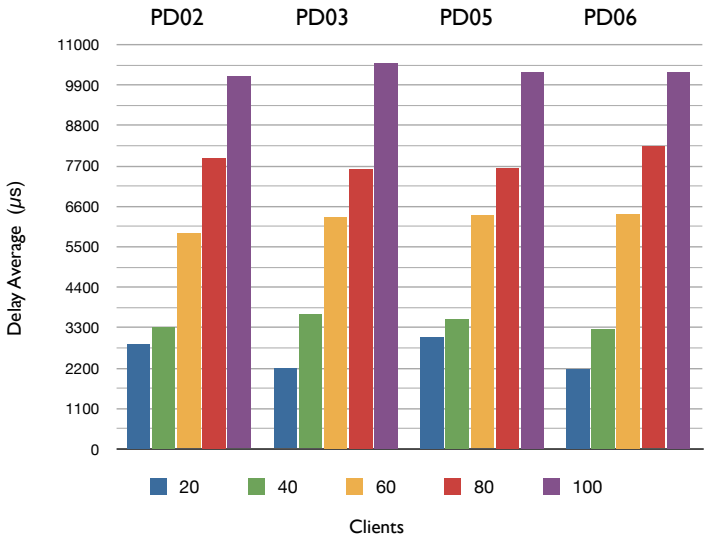


Fig. 5. Scalability of master and multiple slaves topology.

4.3 Results

Results obtained with 100 TPC-C clients and the master and multiple slaves topology are presented in (Table 1). It can be observed that all replicas get similar results and that the propagation delay is consistently measured close to 10 ms with a small variability. This represents an upper bound on the worst case scenario staleness that a client can observe by reading from the master and any other replica if the replication connection is operational.

Results with an different numbers of TPC-C clients can be found in (Figure 5). They show that propagation delay grows substantially with the load imposed on the master. At the same time, as idle periods get less and less frequent due to the higher amount of information to transfer, the probability of a client being able to read stale data grows accordingly.

Results obtained with 100 TPC-C clients and the chain topology are presented in (Table 2). In contrast to master and multiple slaves, the delay now grows as the replica is farther away for the master. This configuration also gives an indication of how the ring topology would perform: As any replica would

Replica	PD02	PD03	PD05	PD06
Number of samples	12423	12819	12937	14004
Average delay (μs)	12353	19767	25698	30688
99% confidence interval ($\pm$)	557	700	864	984

Table 2. Results for chain topology with 100 clients.

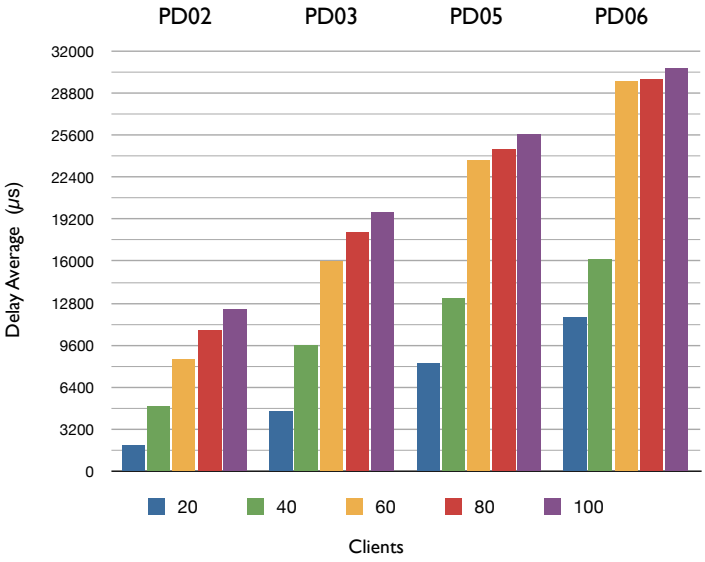


Fig. 6. Scalability of the chain topology.

be, on average, half way to other masters, one should expect the same delay as observed here on replicas PD03 and PD05.

Results with an increasing number of TPC-C clients can also be found in (Figure 6), showing that propagation delay still grow substantially with the load imposed on the master. This means that using the ring configuration for write scalability with suffer the same problem, thus limiting its usefulness.

5 Conclusions

Asynchronous, or lazy, database replication is often the preferred approach for achieving large scale and highly available database management systems. In particular, the replication mechanism in MySQL is at the core of some of the largest databases in use today for Internet applications. In this paper we set out to evaluate the consequences on data freshness of the choice of replication topologies and of a growing workload.

In short, our approach measures freshness in terms of time required for updates performed at the master replica to reach each slave while using a realistic update-intensive workload, as the proposed tool can infer freshness from a small number of samples taken at different points in time at different replicas. Experimental results obtained with this tool show that, in both tested replication topologies, the delay grows with the workload which limits the amount of updates that can be handled by a single replica. Moreover, we can also conclude that in circular replication the delay grows as the number of replicas increases, which means that spreading updates across several replicas does not improve update scalability. Finally, the delay grows also with the number of slaves attached to each master, which means that read scalability can also be achieved only at the expense of data freshness.

The conclusion is that the apparently unlimited scalability of MySQL using a combination of different replication topologies can only be achieved at the expense of an increasing impact in data freshness. The application has thus to explicitly deal with stale data in order to minimize or prevent the user from observing inconsistent results.

References

1. Transaction Processing Performance Council. TPC Benchmark^{TM} C standard specification revision 5.11, February 2010.
2. K. Daudjee and K. Salem. Lazy database replication with ordering guarantees. *Proceedings of the 20th International Conference on Data Engineering (ICDE 2004)*, 30:424–435, 2004.
3. J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, page 173, 1996.
4. B. Kemme. *Database replication for clusters of workstations*. PhD thesis, Technische Wissenschaften ETH Zürich, Zürich, 2000.
5. C. Le Pape, S. Gancarski, and P. Valduriez. Data quality management in a database cluster with lazy replication. *Journal of Digital Information Management (JDIM)*, 3(2), 2005.
6. M. Özsu and P. Valduriez. Distributed and parallel database systems. *ACM Computing Surveys (CSUR)*, 28(1):125–128, 1996.
7. E. Pacitti, P. Minet, and E. Simon. Fast algorithms for maintaining replica consistency in lazy master replicated databases. Research Report RR-3654, INRIA, 1999.
8. B. Schwartz, J. D. Zawodny, D. J. Balling, V. Tkachenko, and P. Zaitsev. *High Performance MySQL: Optimization, Backups, Replication, and More; 2nd ed.* O'Reilly, 2008.

Exploring Fault-tolerance and Reliability in a Peer-to-peer Cycle-sharing Infrastructure

João Paulino *, Paulo Ferreira, and Luís Veiga

INESC-ID/IST
Rua Alves Redol N°9, 1000-029, Lisboa, Portugal
joaopaulino@ist.utl.pt,{paulo.ferreira, luis.veiga}@inesc-id.pt

Abstract. The partitioning of a long running task into smaller tasks that are executed separately in several machines can speed up the execution of a computationally expensive task. This has been explored in Clusters, in Grids and lately in Peer-to-peer systems. However, transposing these ideas from controlled environments (e.g., Clusters and Grids) to public environments (e.g., Peer-to-peer) raises some reliability challenges: will a peer ever return the result of the task that was submitted to it or will it crash? and even if a result is returned, will it be the accurate result of the task or just some random bytes? These challenges demand the introduction of result verification and checkpoint/restart mechanisms to improve the reliability of high performance computing systems in public environments. In this paper we propose and analyse a twofold approach: i) two checkpoint/restart mechanisms to mitigate the volatile nature of the participants; and ii) various flavours of replication schemes for reliable result verification.

Keywords: fault-tolerance, result verification, checkpoint/restart, cycle-sharing, public computing

1 Introduction

The execution of long running applications has always been a challenge. Even with the latest developments of faster hardware, the execution of these is still infeasible by common computers, for it would take months or even years. Even though super-computers could speed up these executions to days or weeks, almost no one can afford them. The idea of executing these in several common machines parallely was firstly explored in controlled environments [19, 3, 9, 4] and was later transposed to public environments [2, 12]. Although they are based on the same principles, new challenges arise from the characteristics of public environments.

Clusters [19, 3] and Grids [9, 4, 13] have been very successful in accelerating computationally intensive tasks. The major difference between these is that

* This work was supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds. João Paulino and this work were supported by FCT research project GINGER - PTDC/EIA/73240/2006

while clusters use dedicated machines in a local network, grids consider the opportunistic use of workstations owned by institutions around the Globe. Both systems are composed by well managed hardware, trusted software and a near 24 hour per day uptime. Public computing [2, 12, 1, 5, 10, 15] stems from the fact that the World's computing power and disk space is no longer exclusively owned by institutions. Instead, it is distributed in the hundreds of millions of personal computers and game consoles belonging to the general public. These systems face new challenges inherent to their characteristics: less reliable hardware, untrusted software and unpredictable uptime.

One of the several public computing projects is GINGER (Grid Infrastructure for Non Grid EnviRonments), in the context of which the work of this paper has been developed. GINGER [20] proposes an approach based on a network of favours where every peer is able to submit his work-units to be executed on other peers and execute work-units submitted by other peers as well. A specific goal of GINGER is that in order to be able to run an interesting variety of applications without modifying them, GINGER proposes the concept of Gridlet, a semantics-aware unit of workload division and computation off-load (basically the data, an estimate of the cost, and the code or a reference to it). Therefore, GINGER is expected to run applications such as audio and video compression, signal processing related to multimedia content (e.g., photo, video and audio enhancement, motion tracking), content adaptation (e.g., transcoding), and intensive calculus for content generation (e.g., ray-tracing, fractal generation).

The highly transient nature of the participants in the system may origin a constant loss of already performed work when a peer fails/leaves or even the never ending of a task, if no peer is ever enough time available to accomplish it. To mitigate this, checkpointing/restart mechanisms shall be able to save the state of a running application to safe storage during the execution. Allowing it to be resumed in another peer from the point when it was saved if necessary.

The participants of the system are not trusted, so are the results they return. Results may be invalid (e.g., either corrupted data or format non-compliance), or otherwise valid but in disagreement with input data (e.g., repeated results from previous executions with different input, especially one computationally lighter). Therefore, result verification mechanisms shall be able to check the correctness of the results.

In the next Section, we address the relevant related work to ours. In Section 3, we propose result verification techniques and checkpoint/restart mechanisms. In Section 4 we provide a description of our implementation. In Section 5, we evaluate the proposed techniques. Section 6 concludes.

2 Related Work

In Section 2.1 we review the main approaches to provide an application with checkpoint/restart capabilities; in Section 2.2 we analyse the techniques that are mainly used to verify the correctness of the results.

2.1 Checkpoint/Restart

Checkpoint/restart is a primordial fault-tolerance technique. Long running applications usually implement checkpoint/restart mechanisms to minimize the loss of already performed work when a fault occurs [16, 8]. Checkpoint consists in saving a program's state to stable storage during fault-free execution. Restart is the ability to resume a program that was previously checkpointed. To provide an application with these capabilities, various approaches have been proposed: Application-level [2, 12] , Library-level [18, 14] and System-level [21].

Application-level Checkpoint/Restart Systems. These systems do not use any operating system support. These are usually more efficient and produce smaller checkpoints. They also have the advantage of being portable[1]. These checkpointing mechanisms are implemented within the application code, requiring a big programming effort. Checkpointing support built in the application is the most efficient, because the programmer knows exactly what must be saved to enable the application to restart. Though, this approach has some drawbacks: it requires major modifications to application's source code (its implementation is not transparent [2] to the application); the application will take checkpoints by itself and there is no way to order the application to checkpoint if needed; it may be hard, if not impossible, to restart an application that was not initially designed to support checkpointing; and it is a very exhaustive task to the programmer. This programming effort can be minimized using pre-processors that add checkpointing code to the application's code, though they usually require the programmer to state what needs to be saved (e.g., through flagged/annotated code). Public computing systems like Seti@home [2] and Folding@home [12] use this checkpointing solution.

Library-level Checkpoint/Restart Systems. This approach consists in linking a library with the application, that creates a layer between the application and the operating system. This layer has no semantic knowledge of the application and cannot access kernel's data structures (e.g., file descriptors), so this layer has to emulate operating system calls. The major advantage is that a portable generic checkpointing mechanism could be created, though it is very hard to implement a generic model to checkpoint any application. This checkpointing method requires none or very few modifications to the application's code. This approach has been explored by libckpt [18] and Condor [14].

System-level Checkpoint/Restart Systems. These systems are built as an extension of the operating system's kernel [21], therefore they can access kernel's data structures. Checkpointing can consist in flushing all the process's data and control structures to stable storage. Since these mechanisms are external to the application they do not require specific knowledge of it, and they require none

[1] Portability is the ability of moving the checkpoint system from one platform to another.

[2] Transparency is the ability of checkpointing an application without modifying it.

or minimal changes to the application. They have the obvious disadvantage of not being portable and usually more inefficient than application-level.

2.2 Result Verification

The results returned by the participants may be wrong due either to failures or malicious behaviour. Failures occasionally produce erroneous results that must be identified. Malicious participants create bad results that are intentionally harder to detect. Their motivation is to discredit the system, or to grow a reputation for work they have not executed (i.e., to cheat the public computing system and exploit other participants' resources).

Replication. One of the most effective methods to identify bad results is through redundant execution. In these schemes the same job is performed by N different participants (N being the replication factor). The results are compared using voting quorums, and if there is a majority the corresponding result is accepted. Since, it is virtually impossible for a fault or independent byzantine behaviour to produce the same bad result more than once, this technique easily identifies and discards the bad ones. However, if a group of participants colludes it may be hard to detect a bad result [17, 6]. Another disadvantage is the massive overhead it generates. Most of the public computing projects [2, 12] use replication to verify their results, it is a high price they are willing to pay to ensure their results are reliable. However, when there is no collusion, it is virtually capable of identifying all the bad results with 100% certainty.

Hash-trees. Cheating participants can be defeated if they are forced to calculate a binary hash-tree from their results, and return it with them [7]. The submitting peer only has to execute a small portion of a job and calculate its hash. Then, when receiving results, the submitting peer compares the hashes and verifies the integrity of the hash-tree. This dissuades cheating participants because finding the correct hash-tree requires more computation than actually performing the required computation and producing the correct results. The leafs of the hash tree are the results we want to check. The hash is calculated using two consecutive parts of the result concatenated, starting by the leafs. Once the tree is complete, the submitting peer executes a random sample of the whole work that corresponds to a leaf. Then this result is compared to the returned result and the hashes of the whole tree are checked. Hash-trees make cheating not worthwhile. They have a relative low overhead: a small portion of the work has to be executed locally and the hash tree must be checked.

Quizzes. Quizzes are jobs whose result is known by the submitter a priori. Therefore, they can test the honesty of a participant. Cluster Computing On the Fly [15] proposed two types of quizzes: stand-alone and embedded quizzes.

Stand-alone quizzes are quizzes disguised as normal jobs. They can test if the executing peer executed the job. These quizzes are only useful when associated with a reputation mechanism that manages the trust levels of the executing peers

[11]. Though, the use of the same quiz more than once can enable malicious peers to identify the quizzes and to fool the reputation mechanisms. The generation of infinite quizzes with known results incurs considerable overhead.

Embedded quizzes are smaller quizzes that are placed hidden into a job, the job result is accepted if the results of the embedded-quizzes match the previously known ones. These can be used without a reputation system. Though, the implementation tends to be complex in most cases. Developing a generic quiz embedder is a software engineering problem that has not been solved so far.

3 Architecture

In section 3.1 we propose two checkpointing mechanisms that enable GINGER to checkpoint and restart any application; in section 3.2 we discuss result verification techniques that we implemented in GINGER, we consider various flavours of replication and a straightforward sampling technique.

3.1 Checkpoint/Restart Mechanisms

In GINGER we want to provide a wide range of applications with checkpoint/restart capabilities, while keeping them portable to be executed on cycle-sharing participant nodes, and without having to modify them. Library-level is the only approach in the related work that would fit. However, an approach simply stating these goals is still far from being able to checkpoint any application. Therefore, we propose two mechanisms that will enable us to checkpoint any application.

Generic Checkpoint/Restart. An application can be checkpointed if we run it on top of virtual machine with checkpoint/restart capabilities (e.g., qemu), being the application state saved within the virtual machine state. This also provides some extra security to the clients, since they can be executing untrusted code. The major drawback of this approach is the size of the checkpoint data, incurring considerable transmission overhead. To attenuate this: 1) we assume that one base-generic running checkpoint image is accessible to all the peers; 2) the applications start their execution on top of this image once it is locally resumed; and 3) at checkpoint time we only transmit the differences between the current image and the base-image. The checkpoint data size can be further reduced using various techniques: optimized operating systems (e.g., just enough operating system or JeOS); differencing not only the disk but also the volatile state; and applying compression to the data. This approach does not have semantic knowledge of the applications, it cannot preview results. However we may be able to show some statistical data related to the execution and highlight where changes have occurred.

Result Checkpoint/Restart. This technique will only be fit for some applications and demands the implementation of specific enabling mechanisms for each application. The idea behind this technique is that the applications produce final results incrementally during their execution. Therefore, if we are able

to capture the partial results during execution and resume execution from them later, such result files can actually serve as checkpoint data. This creates a very efficient checkpointing mechanism. This technique can be implemented using two different approaches: by monitoring the result file that is being produced by the application; or by dividing the gridlet work into subtasks in the executing peer. Since this approach has semantic knowledge of the application result it can checkpoint whenever it is more convenient (e.g., every 10 lines in an image written by a ray-tracer); rather than on a predefined time interval. This awareness of the semantics of the application also enables the monitoring of the execution and the previewing of the results in the submitter.

3.2 Result Verification Mechanisms

In order to accept the results returned by the participants we propose replication with some extra considerations and a complementary sampling technique.

Incremental Replication. The insight of assigning the work iteratively according to some rules, instead of putting the whole job to execution at once can provide some benefits with only minor drawbacks.

The major benefit stems from the fact that lots of redundant execution is not even taken into consideration when the correct result is being chosen by the voting quorums. For example, for replication factor 5, if 3 out of the 5 results are equal the system will not even mind looking at the other 2 results. Then, those could and should have never been executed. And if so, the overall execution power of the system would have been optimized by avoiding useless repeated work. This replication scheme has additional benefits in colluding scenarios. In these, the same bad result is only returned once the colluders have been able to identify that they have the same job to execute. If a task is never being redundantly executed at the same time, colluders can only be successful if they submit a bad result and wait for the replica of that task to be assigned to one of them, enabling them to return the same bad result. If that does not happen, the bad result submitted by them will be detected and they might be punished by an associated reputation mechanism (e.g., blacklisted).

This technique can have a negative impact in terms of time to complete the whole work: in one hand, the incremental assignment and wait for the retrieval of results will lower the performance when the system is not overloaded; on the other hand, if the number of available participants is low it can actually perform faster than putting the whole work to execution at once. Therefore, the correct definition of an overloaded environment having into consideration various factors (e.g., the number of available participants, the maximum number of gridlets, etc.) makes possible for the system to decide whether to use this technique or not, enabling it to take the best advantage of the present resources.

Replication with Overlapped Partitioning. Using overlapped partitioning the tasks are never exactly equal, even though each individual piece of data is still replicated with the predetermined factor. Therefore, it becomes more complex for

the colluders to identify the common part of the task, plus they have to execute part of the task. Figure 1 depicts the same work divided in in two different overlapped partitionings. Overlapped partitioning could be implemented in a

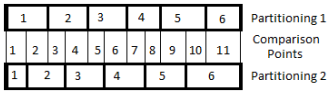


Fig. 1. The same work divided differently creating an overlapped partitioning.

relaxed flavour, where only some parts of the job are executed redundantly. This lowers the overhead, but also lowers the reliability of the results. However, it can be useful if the system has low computational power available. Figure 2 depicts a relaxed overlapped partitioning.

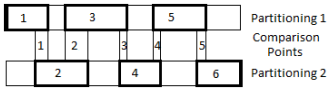


Fig. 2. Overlapped tasks for relaxed replication.

Replication with Meshed Partitioning. Some applications can have their work divided in more than one dimension. Figure 3 depicts the partitioning of the work for a ray-tracer. Like the overlapped partitioning it influences the way colluders are able to introduce bad results: more points where they can collude, with a smaller size too. This partitioning provides lots of points of comparison. This information might feed an algorithm that is able to choose correct results according to the reputation of a result, instead of using voting quorums.

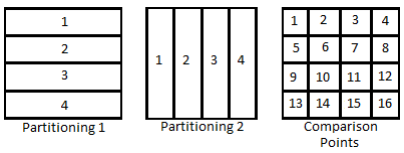


Fig. 3. Meshed partitioning using replication factor 2.

Sampling. Replication bases all its result verification decisions in results/info provided by third parties, i.e., the participant workers. In an unreliable environment this may not be enough. Therefore, local sampling can have an important place in the verification of results. Sampling considers the local execution of a fragment, as small as possible, of each task to be compared with the returned result. In essence, sampling points act as hidden embedded quizzes. This sample is the only trusted result, so even if a result that was accepted by the voting

quorums does not match the local sample it is discarded. Figure 4 depicts the sampling of an image where a sample is a pixel. We have proposed two check-

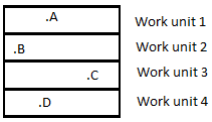


Fig. 4. Sampling for an image.

point/restart enabling techniques. Our generic checkpointing technique enables us to checkpoint any application and resume it later, the overhead it incurs derives from the size of the checkpoint data. Nevertheless, for some applications our result oriented technique will enable the system to checkpoint and resume an application with no noticeable overhead, the results are transmitted incrementally, rather than at the end of the execution. For a reliable result verification we have proposed various flavours of replication that make colluding increasingly more difficult to achieve and easier to detect. Replication is combined with a sampling technique that tests the received untrusted results against one result sample that is known to be correct.

4 Implementation

Our implementation is developed in two different deployments: i) a simulation environment, that enables us to test result verification approaches with large populations; and ii) a real environment, that proves that our result verification and the checkpointing approaches are feasible.

4.1 Simulation Environment

The simulator is a Java application that simulates a scenario where an n-dimensional job is broken into work-units that are randomly assigned. Among the participants there is a group of colluders that attempt to return the same bad result (based on complete or imperfect knowledge, depending on the partition overlapping), in order to fool the replication based verification mechanisms. The simulator receives several parameters: number of participants; number of colluders; work-size as an array of integers (n-dimensional representation), the size as defined in terms of atoms of execution (i.e., an indivisible portion of execution); number of gridlets; replication factor; and partitioning mode (standard or overlapped). The simulator returns several results, being the most important one the percentage of bad results that were accepted by the system.

4.2 Real Environment

For being able to support a new application, semantic knowledge of it is mandatory. Therefore, 3 classes must be programmed: an application manager, a gridlet

and a result. The Application Manager is responsible for translating a command that invokes an application into several executable gridlets and reunite their partial results once it has verified their correctness. For some applications it also enable the user to preview the results (e.g., an image being incrementally produced by a ray tracer). It must extend the abstract class ApplicationManager and implement a constructor and two methods. The following is an excerpt of the Pov Ray's application manager class.

```
class PovRayManager extends ApplicationManager {

  PovRayManager(String command) throws ApplicationManagerException { ... }

  ArrayList<Gridlet> createGridlets(int nGridlets) { ... }

  void submitResult(Result result) { ... }
}
```

The Gridlet class must implement the Serializable and Runnable interfaces, this enables transportation by the Java RMI and allows it to perform a threaded execution in its destination. The following is an excerpt of the Pov Ray's gridlet class.

```
class PovRayGridlet extends Gridlet implements Serializable, Runnable {

    void run() { ... }
}
```

The Result class is just a container of the result data of a gridlet, it must be defined for each application and implement the Serializable interface, for the Java RMI mechanisms being able to transmit it.

5 Evaluation

At this stage of our work, we only have few performance measures of the techniques we have described. We present what we have been able to measure so far in this section.

5.1 Checkpoint/Restart Mechanisms

The major issue of the generic checkpoint/restart technique is transmitting the state of a virtual machine. We are mitigating this by transmitting only the differences between the current image and the base-image. The table in Figure 5 depicts the size of the checkpoint data to be transmitted.

		size (KB)	total size (KB)
Base Image	disk image - powered off (.vdi)	2.650.145,00	2.764.961,00
	disk image - after OS boot (differencial .vdi)	33,00	
	volatile state - after OS boot (.sav)	114.783,00	
Current Image	disk image - running application (differencial .vdi)	21.537,00	161.455,00
	volatile state - running application (.sav)	139.918,00	

Fig. 5. Checkpoint data size using VirtualBox and Ubuntu Desktop 9.10

5.2 Result Verification Mechanisms

Replication can be fooled if a group of colluders determines they are executing redundant work and agree to return the same bad result, forcing the system to accept it. The graphic in Figure 6 depicts that when the percentage of colluders is under 50%, the greater the replication factor the lower the percentage of bad results accepted; when the percentage of colluders is above 50%, albeit a less probable scenario, replication actually works against us. Groups of colluders are usually expected to be minorities. However we must take into account that if they are able to influence the scheduler by announcing themselves as attractive executers the percentage of bad results could even be above what this graphic shows, for the scheduler it uses is random. Overlapped partitioning influences the

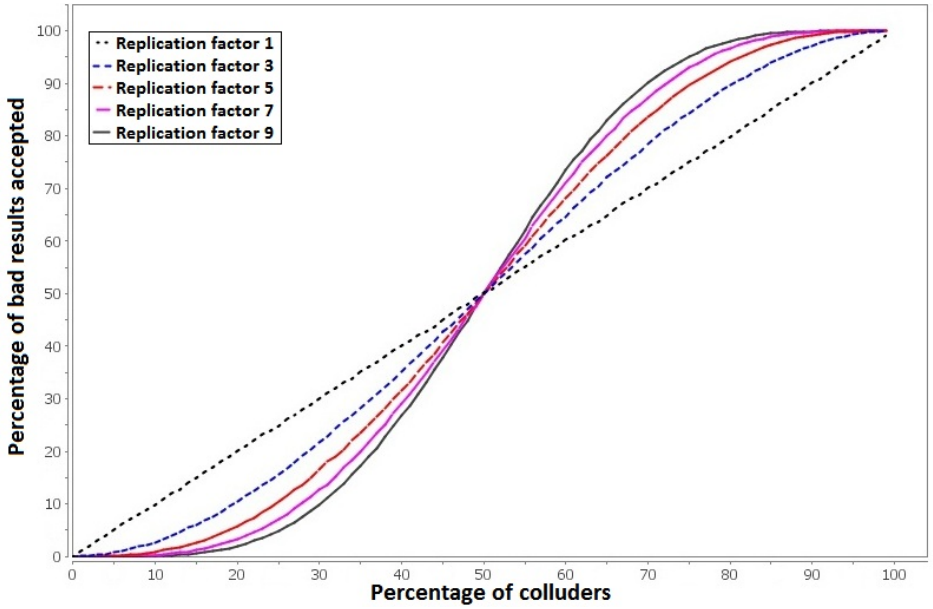


Fig. 6. Correlation between the percentage of bad results accepted and the percentage of colluders in the system for various replication factors.

way that the colluders introduce their bad results: it produces more points where collusion may happen and also may be detected; the size of each bad result is smaller, though. This happens because one task is replicated into more tasks than using standard partitioning; therefore there is a higher probability of redundant work being assigned to colluders; however they can only collude part of the task instead of the whole task as using standard partitioning. The graphic in Figure 7 depicts that overlapped partitioning is as good as standard partitioning, in a scenario where the colluders are fully able to identify the common part (in theory possible, but in practice harder to achieve as this may require global knowledge and impose heavier coordination and matching of information among the colluders) and collude it, while executing the non common part. This is the worst case scenario, therefore overlapped partitionings can improve the reliability of the results depending on how smart the colluders are.

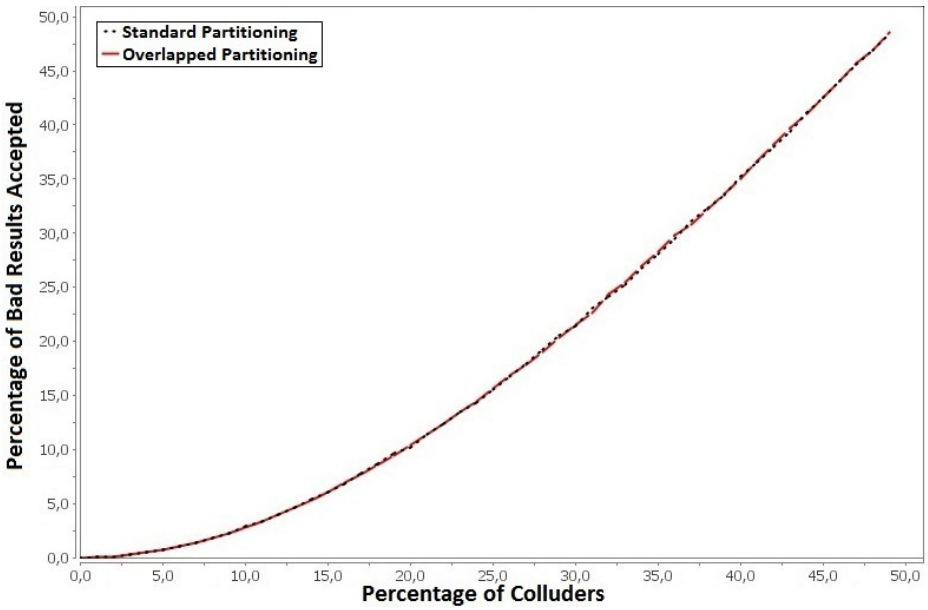


Fig. 7. Replication w/ Standard Partitioning Vs. Replication w/ Overlapped Partitioning, using replication factor 3.

6 Conclusions

In this paper, we proposed and analysed a number of checkpoint/restart mechanisms and replication schemes to improve fault-tolerance and reliability in cycle-sharing infrastructures such as those in Peer-to-peer networks.

Our generic checkpoint/restart mechanism based in virtual machine images that is able to checkpoint any application. It is yet at an early stage of development, we see no obstacles to it other than the checkpoint data size, therefore we are focused in reducing it. Our result oriented approach is only fit for some applications. We have successfully enabled POV-Ray to checkpoint and resume execution from its results automatically.

Our result verification schemes are very promising, we are trying to figure out how can we adapt them to the variable conditions of the system in order to produce a good compromise between performance and reliability.

References

1. D. P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.
2. D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.
3. T. E. Anderson, D. E. Culler, D. A. Patterson, , and the NOW team. A case for now (networks of workstations). *IEEE Micro*, 15:54–64, 1995.

4. L. B. Costa, L. Feitosa, E. Araujo, G. Mendes, R. Coelho, W. Cirne, and D. Fireman. Mygrid: A complete solution for running bag-of-tasks applications. In *In Proc. of the SBRC 2004, Salao de Ferramentas, 22nd Brazilian Symposium on Computer Networks, III Special Tools Session*, 2004.
5. distributed.net. Distributed.net: Node zero. In *http://distributed.net/*, 2010.
6. J. R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.
7. W. Du, J. Jia, M. Mangal, and M. Murugesan. Uncheatable grid computing. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 4–11, 2004.
8. J. C. e Alexandre Sztajnberg. Introdução de um mecanismo de checkpointing e migração em uma infra-estrutura para aplicações distribuídas. In *V Workshop de Sistemas Operacionais (WSO'2008)*, July 2008.
9. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.
10. GIMPS. Great internet mersenne prime search. In *http://mersenne.org*, 2010.
11. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 640–651, New York, NY, USA, 2003. ACM.
12. S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. Technical Report arXiv:0901.0866, Jan 2009.
13. M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.
14. M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. Checkpoint and migration of UNIX processes in the Condor distributed processing system. Technical Report UW-CS-TR-1346, University of Wisconsin - Madison Computer Sciences Department, April 1997.
15. V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster computing on the fly: P2p scheduling of idle cycles in the internet. In *In Proceedings of the IEEE Fourth International Conference on Peer-to-Peer Systems*, pages 227–236, 2004.
16. A. Maloney and A. Goscinski. A survey and review of the current state of rollback-recovery for cluster systems. *Concurr. Comput. : Pract. Exper.*, 21(12):1632–1666, 2009.
17. D. Molnar. The seti@home problem. In *ACM Crossroads: The ACM Student Magazine*, 2000.
18. J. S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent checkpointing under Unix. In *Usenix Winter Technical Conference*, pages 213–223, January 1995.
19. T. Sterling, D. J. Becker, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Packer. Beowulf: A parallel workstation for scientific computation. In *In Proceedings of the 24th International Conference on Parallel Processing*, pages 11–14. CRC Press, 1995.
20. L. Veiga, R. Rodrigues, and P. Ferreira. Gigi: An ocean of gridlets on a 'grid-for-the-masses´. In *IEEE International Symposium on Cluster Computing and the Grid - CCGrid 2007 (PMGC-Workshop on Programming Models for the Grid)*. IEEE Press, May 2007.
21. H. Zhong and J. Nieh. Crak: Linux checkpoint / restart as a kernel module. Technical Report CUCS-014-01, Department of Computer Science. Columbia University., November 2002.

Impacto da Organização dos Dados em Operações com Matrizes Esparsas na GPU

Paula Prata[1,2], Gilberto Melfe[2], Ricardo Pesqueira[2], João Muranho[1,3]

[1] Instituto de Telecomunicações,
[2] Departamento de Informática,
Universidade da Beira Interior, 6201-001 Covilhã, Portugal
[3] IMAR – Instituto do Mar, Departamento de Zoologia, FCTUC,
Universidade de Coimbra, 3004-517 Coimbra
e-mail: pprata@di.ubi.pt, a23049@ubi.pt, ricardopesqueira705@hotmail.com, muranho@mail.telepac.pt,

Resumo. A utilização de placas gráficas (GPU) para operações sobre matrizes, nomeadamente sobre matrizes esparsas, tem sido alvo de intensa investigação. Pretende-se obter o máximo desempenho de uma arquitectura com centenas de cores e um modelo de paralelismo de dados com execução simultânea de milhares de *threads*. Numerosas aplicações científicas e de engenharia manipulam matrizes esparsas, sendo o produto matriz-vector a operação base para vários algoritmos iterativos de resolução de sistemas de equações esparsas. Neste trabalho avaliamos o desempenho da operação matriz-vector para dois dos formatos mais importantes de armazenamento de matrizes esparsas: CSR e ELL. Mostramos como a ordenação das linhas pode aumentar significativamente o desempenho da operação estudada, no caso do formato ELL, e estudamos o comportamento da solução proposta na resolução de sistemas de equações esparsas correspondente a um problema real.

Palavras-Chave: placa gráfica (GPU), CUDA, paralelismo de dados, matrizes esparsas, sistemas de equações lineares, formatos de armazenamento para matrizes esparsas.

1 Introdução

As actuais placas gráficas (Graphics Processing Units – GPU's) permitem obter um desempenho no processamento massivo de dados em vírgula flutuante comparável ao desempenho obtido, até agora, apenas com supercomputadores. O recente aparecimento de interfaces de programação para a GPU como o "Compute Unified Device Architecture" (CUDA) da NVIDIA [1], o Brook+ da AMD/ATI [2] ou ainda o OpenCL [3], inicialmente desenvolvido pela Apple e posteriormente generalizado para outras arquitecturas, tornou possível programar a placa gráfica usando linguagens de alto nível como o C/C++, ou mesmo o Java. A maior facilidade de programação fez com que a GPU começasse a ser usada para sistemas de computação em larga escala. Surgiu assim um aumento significativo da investigação sobre como paralelizar algoritmos já existentes de forma a optimizar a utilização da GPU.

As GPU's são dispositivos com grande capacidade de cálculo mas que não possuem caches que permitam optimizar os acessos à memória. Possuindo uma elevada latência no acesso à memória global, torna-se necessária a execução de milhares de threads de muito baixa granularidade para conseguir tirar partido da GPU. É também necessário distribuir de forma adequada a carga de trabalho pelos conjuntos de threads que constituem as unidades de escalonamento (warps em linguagem CUDA). Cada warp só é dado por concluído quando todas as suas threads (actualmente 32) tiverem terminado.

Nem todos os algoritmos são pois adequados para paralelizar em GPU. A GPU é especialmente útil para aplicações com grande intensidade de cálculo numérico e com paralelismo de dados, onde cálculos similares são executados em grandes quantidades de dados organizados de forma regular (por exemplo, vectores e matrizes).

Vários estudos mostram que, em problemas que manipulam matrizes densas, a GPU permite obter elevados desempenhos [4], [5]. No entanto, enquanto uma matriz densa tem uma estrutura regular, as matrizes esparsas podem ser representadas em formatos bastante irregulares que poderão condicionar o desempenho.

Neste trabalho analisamos a multiplicação matriz-vector em que a matriz é esparsa. Esta operação, pelo número de vezes que é executada, é a operação dominante em vários algoritmos iterativos para resolução de sistemas de equações lineares e em problemas de cálculo de valores próprios. Um estudo recente sobre a implementação do produto matiz esparsa / vector é apresentado em [6] e [7]. Nestes estudos, Bell e Garland mostram que o melhor desempenho para a operação estudada é obtido com o formato de armazenamento ELL (ELLPACK/ITPACK) em matrizes em que o número de elementos não zero por linha varia pouco. Este formato permite que os valores da matriz a processar por um *warp* estejam em posições contínuas de memória, optimizando assim os tempos de acesso. Se as linhas manipuladas por cada *warp* tiverem aproximadamente o mesmo tamanho (isto é, o mesmo número de elementos não zero) então todas as threads estarão ocupadas em simultâneo sem desperdício de capacidade de cálculo. Como, em problemas reais, o número de elementos não zero por linha é variável, propomos a ordenação das linhas de forma a uniformizar o trabalho a realizar por cada *warp* mesmo em matrizes onde os comprimentos das linhas variam substancialmente. Analisamos o impacto da ordenação das linhas para o formato ELL, concluindo que permite obter melhorias de desempenho que podem atingir os 30%. Aplicámos o mesmo mecanismo a um dos formatos de representação de matrizes esparsas mais comum, o CSR (Compressed Sparse Row). Também aqui é obtido algum ganho mas menos significativo, no melhor caso obteve-se um ganho de 13%.

Finalmente seguimos a mesma abordagem para um problema que envolve a resolução de sistemas de equações esparsas através do método do gradiente conjugado. Trata-se de um método iterativo, que executa a operação matriz-vector em cada uma suas iterações. Os sistemas de equações usados foram gerados por um simulador de redes de distribuição de água, e correspondem a problemas que representam situações reais ou casos de estudo. Concluímos que para estes sistemas de equações, onde as matrizes de coeficientes apresentam um número muito reduzido de valores não zero (abaixo dos 0.5%) e uma certa uniformidade no número de elementos não nulos por linha, o impacto da ordenação das linhas não é substantivo.

Neste artigo apresentamos, na secção 2, o modelo de programação CUDA da NVIDIA e o seu mapeamento na arquitectura da GPU. Na secção 3, descrevemos os formatos de armazenamento de matrizes esparsas usados e as operações implementados. Na secção 4 são apresentados os resultados obtidos e finalmente na secção 5 apresentam-se as conclusões e algumas propostas de trabalho futuro.

2 O Modelo de Programação CUDA e a Arquitectura da GPU

O CUDA veio proporcionar aos utilizadores de linguagens de alto nível, como o C/C++, a possibilidade de tirarem partido do poder de processamento paralelo da GPU [1]. Nesta secção vamos descrever brevemente o modelo de programação CUDA e a arquitectura da GPU usada para este trabalho, a GeForce GTX 295.

2.1 O Modelo de Programação CUDA

No modelo de programação paralela CUDA, uma aplicação consiste na execução de um programa sequencial que corre no CPU (*host*) capaz de lançar na GPU (*device*) funções que serão executadas em paralelo. Cada função que vai ser executada na GPU, designada por *kernel* é executado sequencialmente, mas em simultâneo, por um elevado número de threads. Cada thread tem um ID único podendo ser-lhe atribuída uma tarefa em particular. Há que ter em conta que, em operações que necessitem de pontos de sincronização entre threads, por exemplo quando várias threads manipulam dados em comum, irá haver uma degradação do desempenho. As threads são organizadas em blocos e o conjunto dos blocos de threads irá constituir a grelha de execução (Grid). Cada bloco da grelha de execução irá ser mapeado num multiprocessador da GPU, cuja estrutura descreveremos adiante. Ao ser invocado um *Kerne*l, são passados dois parâmetros especiais que definem o número de threads por bloco e o número de blocos por grelha. Assim sendo, o número de threads que vão executar um *kernel* é o resultado da multiplicação do número de threads de um bloco pelo número de blocos que constitui a grelha de execução.

2.2 A Arquitectura da GPU da NVIDIA

A GPU da NVIDIA é construída como um array de multi-processadores em que cada multi-processador possui 8 núcleos/processadores, um conjunto de registos e uma área de memória partilhada [1]. As operações com valores inteiros e valores em vírgula flutuante de precisão simples são executadas pelos núcleos, enquanto as operações em vírgula flutuante de precisão dupla são executadas por uma unidade partilhada pelos 8 núcleos de um mesmo multiprocessador (apenas para placas com capacidade de computação de 1.3 ou superior).

Quando um *kernel* é lançado, os blocos contidos na grelha de execução são distribuídos e numerados de forma automática para os multiprocessadores com capacidade de os executar. Uma vez associado um bloco de threads a um multiprocessador, o mesmo fica responsável por distribuir as diferentes threads do

bloco pelos diferentes processadores que possui. As threads de um mesmo bloco podem comunicar entre si através da memória partilhada do multi-processador. Uma vez terminado um bloco, e se ainda se encontrarem blocos por executar, os mesmos são automaticamente lançados para um dos multiprocessadores que se encontre livre. Desta forma, pode dizer-se que uma thread está associada a um processador, um bloco a um multi-processador e para cada *kernel* é definida uma grelha (ver figura 1).

Devido ao facto de a GPU apenas tratar dados armazenados na sua memória, os mesmos têm de ser copiados para a memória global da GPU antes da execução do *kernel*.

A GPU usada neste trabalho, a GeForce GTX 295, possui 30 multiprocessadores, cada um com 8 cores, isto é um total de 240 cores (a 1.24GHz), suporta até 512x512x24 blocos com um máximo de 512 threads por bloco. Esta GPU tem 2GB de memória global e capacidade de computação 1.3. Foi programada usando a versão 2.3 do CUDA. A máquina hospedeira é um Intel Core 2 Quad Q9550 a 2.83 GHz com 4 GB de RAM, com sistema operativo Microsoft Windows XP Professional 64-bit.

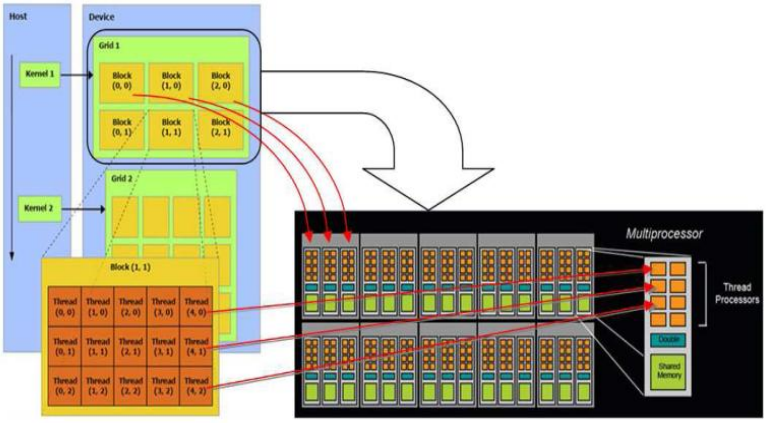


Fig. 1. Associação entre a estrutura do software e a arquitectura do hardware

3 Matrizes Esparsas

Uma matriz esparsa é por definição uma matriz em que a maioria dos seus elementos é igual a zero. Estas matrizes são geralmente armazenadas de forma compacta de modo a guardar apenas os valores diferentes de zero e sua localização. Obtém-se assim um ganho significativo em termos da memória necessária para o seu armazenamento. Vamos descrever os formatos COO, CSR e ELL e as operações estudadas. Uma representação visual destes formatos pode ser analisada em [6].

3.1 Formatos de Representação

O formato base de representação de matrizes esparsas, muitas vezes usado para posterior conversão para outros formatos, é o formato de coordenadas (COOrdinate

Format) designado por COO. O formato COO consiste em armazenar a matriz em três vectores todos de tamanho igual ao número de elementos não zero da matriz: o vector das linhas onde ficam armazenados os índices das linhas em que cada elemento se encontra, o vector das colunas onde ficam armazenados os índices das colunas em que cada elemento se encontra e o vector de dados onde são armazenados os valores dos elementos não zero. Neste trabalho, este formato foi usado para armazenar as matrizes geradas, sendo também o formato usado no repositório de matrizes Matrix Market [8] utilizado como fonte de matrizes de teste.

Formato de Compressão Segundo Linhas. O Formato de compressão segundo linhas ou CSR pode ser considerado como uma extensão do formato de coordenados. A diferença do formato de compressão CSR para o formato COO consiste na substituição do vector que contém os índices das linhas por um outro, geralmente mais curto, de apontadores para a posição no vector das colunas do índice do primeiro elemento não zero da linha correspondente à posição do primeiro vector (isto é, do vector de apontadores). O tamanho do novo vector é o número de linhas mais um, sendo o último elemento do vector o número de não zeros existentes na matriz. Através da subtracção do elemento da posição i+1 pelo elemento da posição i, obtém-se o número de elementos que contém a linha i. O vector de dados vai conter apenas os elementos não zero, ordenados por linhas.

Formato ELLPACK/ITPACK. O formato ELLPACK/ITPACK ou ELL utiliza dois vectores, um para os valores não zero e um outro para os índices das colunas. Supondo uma matriz com M linhas e em que K é o número máximo de elementos não zero por linha, os valores não zero são armazenados por colunas considerando que cada linha tem comprimento K. Para linhas com o número de não zeros menor que K, as posições finais serão preenchidas com um valor pré-definido, por exemplo com o valor zero. Assim, os primeiros M elementos do vector de dados serão o primeiro elemento não zero da primeira linha, o primeiro elemento não zero da segunda linha e assim sucessivamente até ao primeiro elemento não zero da linha M. O segundo vector, o vector dos índices das colunas, corresponde a uma matriz de dimensão M por K, armazenada por colunas, em que cada posição representa a coluna do valor na posição correspondente no vector de dados. Os algoritmos de manipulação deste formato, utilizados neste trabalho, foram optimizados através da utilização de um terceiro vector, consistindo do número de não zeros da linha correspondente.

3.2 Operações estudadas

A operação matriz-vector, como operação base do produto de matrizes, surge nos algoritmos de factorização de matrizes (factorização LU, Cholesky, QR, etc) e portanto, nos algoritmos de resolução de sistemas, algoritmos de cálculo de valores e vectores próprios, algoritmos de decomposição em valores singulares, entre outros. É pois uma operação que apesar de muito simples, sendo executada milhões de vezes, pode ser crítica no desempenho de numerosas aplicações.

Produto Matriz Esparsa/Vector em GPU. No trabalho apresentado em [6] são estudadas três implementações desta operação em GPU: atribuir uma thread a cada elemento não zero da matriz, atribuir uma thread a cada linha da matriz e finalmente atribuir um *warp* a cada linha da matriz. Os resultados mostram que os melhores desempenhos são obtidos para o formato ELL quando cada linha da matriz é processada por uma thread e para o formato CSR quando cada linha é processada por um *warp*. Neste trabalho usámos os formatos ELL e CSR como formatos base para o estudo do efeito da ordenação das linhas da matriz no desempenho da operação.

No formato ELL, como a matriz está armazenada por colunas, a atribuição de uma thread por linha, com a thread i a processar a linha i, acedendo na iteração j ao j-ésimo elemento da linha i, permite que as threads de um mesmo *warp* acedam em cada iteração a posições de memória contíguas, na iteração j, cada thread i acede ao j-ésimo elemento da sua linha. Na iteração 1, a thread 1 vai processar o primeiro elemento da linha 1, a thread 2 o primeiro elemento da linha 2, e assim por diante. Sendo a matriz armazenada por colunas, todos estes elementos estão em posições consecutivas de memória, e uma vez que as threads do mesmo *warp* executam em cada instante a mesma instrução, temos as threads a aceder a posições consecutivas da memória. Este padrão de acesso à memória global, conhecido em linguagem CUDA por "coalesced memory access", é o que permite o melhor desempenho [9].

Se as matrizes tiverem linhas com um número muito variável de elementos não zero, irá acontecer que num mesmo *warp* haverá threads com muito trabalho, isto é com linhas com muitos elementos e threads com pouco trabalho, isto é, com linhas com poucos elementos. Como em cada processador o modelo de execução é SIMD (Simple Instruction Multiple Data) o *warp* só terminará quando todas as suas threads terminarem. É assim de esperar que a ordenação das linhas pelo seu comprimento consiga uniformizar o trabalho de cada *warp* e melhorar o desempenho. Note-se que a utilização de um vector com os tamanhos de cada linha permite parar o processamento no momento adequado (quando a thread já não tem mais elementos para processar) sem alterar o padrão de acesso à memória (mesmo que uma thread ou mais threads não acedam à memória não é violado o esquema dos acessos ordenados).

No caso do formato CSR, o melhor desempenho é obtido colocando as 32 threads de um *warp* a processar a mesma linha. Para os multiprocessadores o modelo de execução é SPMD (Simple Program Multiple Data) e portanto quando um *warp* termina é lançado novo *warp*. Assim, se as threads do mesmo *warp* estão a processar elementos da mesma linha da matriz, a troca de linhas não poderá trazer qualquer ganho, donde neste formato optámos por estudar o impacto da ordenação das linhas só para a variante de uma thread por linha.

Resolução de Sistemas de Equações em GPU. Um problema em que a operação matriz-vector é utilizada frequentemente é a resolução de sistemas de equações, nomeadamente em algoritmos iterativos para sistemas de equações esparsas [10].

Neste trabalho usamos os *kernels* anteriores para matrizes esparsas, em formato CSR e ELL, para paralelizar o produto matriz-vector em cada iteração do método do gradiente conjugado na resolução do sistema de equações esparsas associadas a um programa de simulação de sistemas de abastecimento de água sob pressão, o EPANET [11]. Para cada um dos formatos, estudámos o impacto da ordenação das linhas no desempenho do algoritmo.

4 Resultados

Para avaliarmos o desempenho da operação matriz-vector, gerámos matrizes esparsas quadradas de ordem 4096, 8192 e 16384 contendo valores aleatórios. Para cada linha foi gerado um valor aleatório entre 1 e 20% do número de colunas da matriz. Este será o número de valores não zero da linha, a gerar. A posição de cada valor não zero foi também gerada aleatoriamente. Foram geradas matrizes com valores em precisão simples (*float*) e precisão dupla (*double*). No final, as matrizes geradas tinham 10,01% de valores não zero.

Na figura 2 mostra-se o gráfico do número de valores não zero por linha para a matriz de ordem 4096. Como se pode observar existe variabilidade no comprimento das linhas. Entenda-se por comprimento da linha o número de elementos não zero. Pela forma como foram geradas, as outras matrizes têm gráficos de distribuição semelhantes.

A execução da operação matriz-vector em CPU para os formatos CSR e ELL mostrou que o formato ELL é bastante mais lento que o formato CSR, o que se deve ao armazenamento da matriz por colunas, para um algoritmo que em CPU percorre a matriz por linhas. Os tempos médios, obtidos com 4 execuções, em milissegundos são mostrados nas tabelas 1 e 2 respectivamente para formato CSR e formato ELL, na linha correspondente à ordem da matriz e à frente do tipo de valores. Por razões de falta de espaço não pôde ser incluída uma tabela independente. Estes valores servirão para comparação com os valores obtidos em GPU. Em todos os resultados os tempos de GPU representam o tempo de execução do *kernel* não incluindo a cópia dos dados para a memória da GPU nem a cópia dos resultados para o *host*. Na obtenção dos resultados com ordenação das linhas o tempo gasto na ordenação (feita em CPU) não é considerado. Repare-se que geralmente os problemas reais envolvem um grande número de iterações, sendo a ordenação das linhas realizada uma única vez.

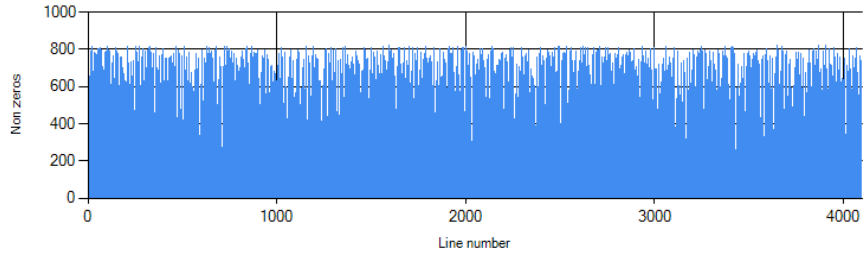


Fig. 2. Gráfico com número de não zeros (non zeros) por linha (line number).

4.1 Produto Matrix-Vector em GPU sem Ordenação de Linhas

Executámos para as mesmas matrizes o produto matriz-vector em GPU formatos CSR e ELL fazendo variar o tamanho do bloco de threads. Considerámos blocos com 32, 64 e 128 threads. O tamanho da grelha, isto é, o número de blocos é obtido dividindo a ordem da matriz pelo tamanho do bloco.

A tabela 1 mostra, além dos tempos em CPU que já referimos, os tempos de execução obtidos com o formato CSR em GPU por matriz e tamanho do bloco. Mostra ainda a

percentagem do tempo de execução em relação ao tempo de CPU, isto é, se houve ou não ganho com a execução em GPU. A tabela 2 mostra os resultados correspondentes para ELL.

Analisando os resultados, podemos ver que o formato CSR em GPU, com uma thread por linha não tem qualquer ganho em relação ao CPU. Para as matrizes mais pequenas parece haver algum ganho mas temos de ter em conta que não contabilizamos o tempo de cópia dos dados. Em precisão dupla, nunca há qualquer ganho o que se explica pelo facto de a unidade de dupla precisão ser partilhada pelos 8 cores de um mesmo multiprocessador. Para o formato ELL em GPU os resultados são completamente diferentes. O tempo gasto em GPU é sempre menos de 5% do tempo gasto em CPU. Observamos que quanto maior a matriz, maior o ganho em relação ao CPU e que quanto maior o tamanho do bloco, isto é quanto mais threads por bloco, melhor desempenho. Concluímos que o formato ELL permite uma utilização efectiva da capacidade de cálculo da GPU.

Tabela 1. Produto matriz-vector em GPU, formato CSR, sem ordenação de linhas

Bloco	**Grelha**	**Tempo (ms)**	**GPU/CPU*100**	**Tempo (ms)**	**GPU/CPU*100**
Ordem: 4096		Float *(CPU: 5.613)*		Double *(CPU: 3.688)*	
32x1	128x1	4.237	75.49	4.734	128.48
64x1	64x1	4.270	76.07	4.794	129.98
128x1	32x1	4.288	76.39	4,855	131.62
Ordem: 8192		Float *(CPU: 22.489)*		Double *(CPU: 15.849)*	
32x1	256x1	20.692	92.01	22.780	143.73
64x1	128x1	20.286	90.20	22.324	140.85
128x1	64x1	20.221	89.91	22.452	142.66
Ordem: 16384		Float *(CPU: 90.212)*		Double *(CPU: 63.105)*	
32x1	512x1	94.238	104.46	101.237	160.42
64x1	256x1	94.019	104.22	101.228	160.41
128x1	128x1	93.371	103.50	100.663	159.52

Tabela 2. Produto matriz-vector em GPU, formato ELL, sem ordenação de linhas

Bloco	**Grelha**	**Tempo (ms)**	**GPU/CPU*100**	**Tempo (ms)**	**GPU/CPU*100**
Ordem: 4096		Float *(CPU: 29.985)*		Double *(CPU: 30.347)*	
32x1	128x1	1.404	4.68	1.440	4.74
64x1	64x1	1.408	4.69	1.463	4.82
128x1	32x1	1.402	4.67	1.486	4.89
Ordem: 8192		Float *(CPU: 197.542)*		Double *(CPU: 147.589)*	
32x1	256x1	5.874	2.97	6.342	4.29
64x1	128x1	4.643	2.35	5.033	3.41
128x1	64x1	4.715	2.38	5.155	3.49
Ordem: 16384		Float *(CPU: 958.678)*		Double *(CPU: 929.724)*	
32x1	512x1	19.656	2.05	21.375	2.31
64x1	256x1	19.635	2.05	21.277	2.29
128x1	128x1	17.555	1.83	19.381	2.09

4.2 Produto Matrix-Vector em GPU com Ordenação de Linhas

Para avaliar o impacto da ordenação das linhas fez-se um pré-processamento dos dados de modo a que as linhas ficassem organizadas em memória por ordem crescente do seu tamanho. Para garantir a consistência dos resultados criou-se um vector adicional contendo a posição inicial das linhas.

A tabela 3 mostra os tempos de execução em GPU obtidos para o formato CSR por matriz e tamanho do bloco de threads, com ordenação por tamanho das linhas. Mostra tal como nas tabelas anteriores o ganho em relação à execução em CPU (*GPUor/CPU *100*) e apresenta também a percentagem de tempo de execução em relação ao tempo de GPU sem ordenação, (*GPUor/GPU*100*). *GPUor* representa o tempo em GPU da versão com ordenação. A tabela 4 mostra os resultados correspondentes para ELL.

Tabela 3. Produto matriz-vector em GPU, formato CSR, com ordenação de linhas

Bloco	**Grelha**	**Tempo (ms)**	**GPUor/CPU *100**	**GPUor/GPU *100**	**Tempo (ms)**	**GPUor/CPU *100**	**GPUor/GPU *100**
Ordem: 4096		Float			Double		
32x1	128x1	3.728	66.41	87.97	4.429	120.08	93.47
64x1	64x1	3.731	66.47	87.38	4.440	120.38	92.61
128x1	32x1	3.723	66.32	86.82	4.433	120.20	91.32
Ordem: 8192		Float			Double		
32x1	128x1	18.810	83.64	90.90	21.023	132.65	92.29
64x1	64x1	18.799	83.59	92.67	21.001	132.51	94.07
128x1	32x1	18.852	83.83	93.23	21.028	132.68	93.66
Ordem: 16384		Float			Double		
32x1	128x1	88.587	98.20	94.00	95.437	151.24	94.27
64x1	64x1	88.151	97.72	93.76	94.704	150.07	93.55
128x1	32x1	88.149	97.71	94.41	94.489	149.75	93.88

Tabela 4. Produto matriz-vector em GPU, formato ELL, com ordenação de linhas

Bloco	**Grelha**	**Tempo (ms)**	**GPU/CPU *100**	**GPUor/GPU *100**	**Tempo (ms)**	**GPU/CPU *100**	**GPUor/GPU *100**
Ordem: 4096		Float			Double		
32x1	128x1	1.231	4.11	87.68	1.315	4.33	91.32
64x1	64x1	1.227	4.09	87.16	1.335	4.39	91.23
128x1	32x1	1.228	4.09	87.57	1.363	4.49	91.74
Ordem: 8192		Float			Double		
32x1	128x1	3.948	2.00	67.21	4.365	2.95	68.84
64x1	64x1	3.890	1.97	83.79	4.332	2.94	86.07
128x1	32x1	3.944	2.00	83.65	4.440	3.01	86.13
Ordem: 16384		Float			Double		
32x1	128x1	14.110	1.47	71.78	15.556	1.68	72.78
64x1	64x1	14.931	1.56	76.04	15.837	1.71	74.43
128x1	32x1	14.975	1.56	85.30	16.08	1.74	83.01

Analisando os resultados podemos ver que para o formato CSR há um pequeno ganho quer em relação à execução em CPU quer em relação à execução em GPU sem ordenação. Esse ganho é na generalidade inferior a 10%. Apenas para a matriz de menor ordem existe um ganho de 13% em relação à execução em GPU sem ordenação com um bloco de 128 threads.

Para o formato ELL os resultados são bem melhores. Em precisão simples o ganho em GPU com a ordenação varia entre os 12.32% (para a matriz de ordem 4092, com 32 threads por bloco) e os 33.8% (para a matriz de ordem 8192 com 32 threads por bloco) Em precisão dupla atingem-se também ganhos com a ordenação, à volta dos 30% para as matrizes de maior ordem. Na generalidade dos casos quanto maior a dimensão maior o ganho em GPU com a ordenação por tamanhos das linhas.

4.3 Resolução de Sistemas de Equações

Finalmente estudamos o impacto da ordenação das linhas quando a operação matriz-vector é utilizada na resolução de sistemas de equações no âmbito de uma ferramenta de modelação de sistemas de abastecimento de água sob pressão. As matrizes produzidas apresentam uma percentagem de elementos não zero muito baixa. Para cada problema simulado, produziram-se 3 matrizes correspondendo a diferentes passos da simulação. Os resultados apresentados nas tabelas que se seguem correspondem, para cada tipo de matriz, à média obtida com a execução desse conjunto de matrizes. A tabela 5 mostra, para as matrizes estudadas, o tempo médio de execução do produto matriz-vector em CPU para os formatos CSR e ELL. Apenas é estudado o caso da precisão dupla (double) porque o algoritmo não converge em precisão simples. Como se pode observar, o tempo de execução em ELL é superior ao do formato CSR, tal como anteriormente.

Tabela 5. Produto matriz-vector em CPU para as matrizes dos sistemas de equações.

Matriz	Ordem	Nº de não zeros	% de não zeros	Tempo médio de execução (ms)	
				CSR - double	ELL - double
Richmond	865	3589	0.48	0.018	0.028
Wolf	1782	8244	0.26	0.038	0.054
Exnet	1891	10025	0.28	0.043	0.070
BWSN	12523	62463	0.04	0.360	0.670
FinMar	14991	71001	0.03	0.343	0,680

Na figura 3 podemos observar o gráfico do número de valores não zero por linha para a matriz de maior ordem (*FinMar*). Como se pode observar a estrutura desta matriz é completamente diferente das estudadas anteriormente. Existem algumas linhas com muitos não zeros e depois muitas linhas com 3, 4 ou menos elementos. As outras matrizes da tabela 5 têm estruturas semelhantes.

A tabela 7 apresenta os resultados da resolução do sistema Ax=b em CPU para os formatos CSR e ELL. Para cada tipo de matriz mostra-se o número médio de iterações realizado, o tempo médio de execução e o tempo médio por iteração.

A tabela 8 apresenta os resultados para a resolução em GPU, formato CSR com e sem ordenação. Em GPU foi usado um bloco de 32 threads para ambos os formatos.

Observando os resultados podemos concluir que o formato CSR em GPU tem algum ganho em relação à execução em CPU para as matrizes maiores (BWSN e FinMar) mas que para estas matrizes o impacto da ordenação é praticamente nulo. A ordenação apenas tem um pequeno impacto para as matrizes de menor dimensão mas para estas matrizes a resolução em CPU é mais rápida.

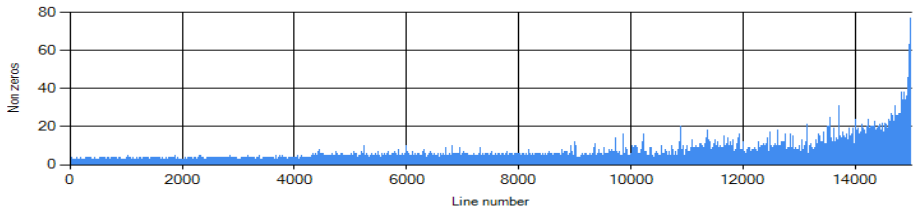


Fig. 3. Gráfico com número de não zeros (*non zeros*) por linha (*line number*) da matriz FinMar

Tabela 7. Resolução em CPU do sistema Ax=b, formatos CSR e ELL.

Matriz A	Nº médio de iterações	Tempo médio (ms) CSR		Tempo médio (ms) ELL	
		Resolução	Por iteração	Resolução	Por iteração
Richmond	901316	16227.926	0.019	20217.685	0.023
Wolf	50311	1953.901	0.059	2357.974	0.071
Exnet	108958	5290.277	0.049	6608.656	0.062
BWSN	844635	290462.537	0.437	357666.410	0.538
FinMar	2362729	903249.271	0.378	1092347.806	0.457

Tabela 8. Resolução em GPU do sistema Ax=b, formato CSR, com e sem ordenação.

Matriz A	Nº médio de iterações	Tempo médio de execução no formato CSR (ms)			
		Sem ordenação		Com ordenação	
		Resolução	Por iteração	Resolução	Por iteração
Richmond	895098	138485.976	0.155	136084.721	0.152
Wolf	50004	8197.074	0.164	8011.004	0.160
Exnet	109346	17922.707	0.164	17134.051	0.157
BWSN	839809	212168.191	0.253	212145.219	0.253
FinMar	2359948	598281.605	0.254	607690.063	0.258

Tabela 9. Resolução em GPU do sistema Ax=b, formato ELL, com e sem ordenação.

Matriz A	Nº médio de iterações	Tempo médio de execução no formato ELL (ms)			
		Sem ordenação		Com ordenação	
		Resolução	Por iteração	Resolução	Por iteração
Richmond	895098	139566.167	0.156	138967.563	0.155
Wolf	50004	7958.645	0.159	7972.861	0.159
Exnet	109346	17401.206	0.159	17217.717	0.157
BWSN	839809	187300.657	0.223	190242.438	0.227
FinMar	2359948	533866.135	0.226	536296.084	0.227

Finalmente a tabela 9 mostra os resultados para a execução do sistema em GPU com as matrizes em formato ELL com e sem ordenação. Podemos concluir que para as matrizes de maior dimensão o formato ELL tem melhor desempenho que o formato CSR mas o impacto da ordenação é nulo. O facto de a ordenação não ter impacto resulta da estrutura da matriz. Como vimos pelo gráfico da figura 3 a variabilidade do comprimento das linhas da matriz (dentro de cada *warp*) é muito pequena e portanto a ordenação das linhas implica alterações sem significado na estrutura das matrizes.

5 Conclusões e Trabalho Futuro

Neste trabalho estudamos o desempenho da operação matriz-vector em GPU para dois formatos de representação de matrizes CSR e ELL. Considerando a resolução em GPU em que uma thread é atribuída a cada linha da matriz, estudámos o impacto da ordenação das linhas pelo seu comprimento, isto é, pelo número de valores não zero. Concluímos que com o algoritmo estudado para GPU o formato CSR tem pouca ou nenhuma vantagem em relação ao CPU, enquanto o formato ELL apresenta ganhos entre os 95 a 98%. O impacto da ordenação para matrizes com cerca de 10% de não zeros e variabilidade no tamanho das linhas é menos de 10% para o formato CSR mas para o formato ELL pode significar ganhos na ordem dos 30%. Estudámos ainda a mesma operação, para os mesmos formatos com e sem ordenação, quando utilizada num algoritmo iterativo de resolução de um sistema de equações. Concluímos que para matrizes com um número de não zeros muito reduzido e pouca variabilidade no comprimento das linhas o impacto da ordenação é nulo.

Como trabalho futuro pretendemos explorar formas de distribuição dos dados com utilização da memória partilhada, mais pequena que a memória global mas de acesso muito mais rápido e estudar o produto matriz-vector para problemas reais que envolvam matrizes com diferentes estruturas.

References

1. NVIDIA Corporation, "NVIDIA CUDA Programming guide", version 2.3.2 (2009).
2. ATI, "Stream Computing – Technical Overview", http://developer.amd.com/gpu/atistreamsdk/pages/default.aspx, acedido em Junho de (2010)
3. Khronos group, OpenCl "Parallel Computing for Heterogeneous Devices", 54 páginas, http://www.khronos.org/opencl/, acedido em Junho de (2010).
4. Volkov, V. and Demmel, J. W., "Benchmarking, GPUs to tune dense linear algebra" in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (Austin, Texas, November 15 - 21, 2008). Conference on High Performance Networking and Computing. IEEE Press, Piscataway, NJ, 1-11.
5. Barrachina, S., Castillo, M. Igual, F.D., Mayo, R. and Qintana-Ortí, "Solving dense linear systems on graphics processors" in Proc. 14th Int'l Euro-Par Conference, volume 5168 of Lecture Notes in Computer Science, pages 739-748, Springer, Aug.2008.
6. Bell, Nathan and Garland, Michael. "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors" in Proc. Supercomputing '09, November 2009.
7. Bell, Nathan and Garland, Michael, "Efficient Sparse Matrix-Vector Multiplication on CUDA". Technical Report NVR-2008-004, Dec. 2008 NVIDIA Corporation.
8. The Matrix Market, http://math.nist.gov/MatrixMarket/, acedido em Maio de 2010.
9. Kirk, David B. and Hwu, Wen-mei W., "Programming Massively Parallel Processors", 258 páginas. Morgan Kaufmann, Elsevier, 2010.
10. Saad, Y. Iterative Methods for Sparse Linear Systems. 2nd Edition. Society for Industrial and Applied Mathematics. 2003.
11. Rossman, L.A., "EPANET Users Manual" Risk Management Research Lab., U.S. Environmental Protection Agency, Cincinnati, Ohio, 2000.

Scalable and Efficient Discovery of Resources, Applications, and Services in P2P Grids*

Raoul Felix, Paulo Ferreira, and Luís Veiga

INESC ID /IST
Rua Alves Redol 9
Portugal

Abstract. Distributed computing enables us to harness all the resources and computing power of the millions of computers connected to the Internet. Therefore, this work describes the ongoing effort to create an efficient and scalable resource discovery mechanism, capable of searching not only for physical resources (e.g. CPU, Memory, etc.), but also services (e.g. facial recognition, high-resolution rendering, etc.) and applications (e.g. ffmpeg video encoder, programming language compilers, etc.) from computers connected to the same Peer-to-Peer Grid network. This is done in a novel way by combining all resource information into Attenuated Bloom Filters, which also allows us to efficiently route messages in a completely decentralized unstructured P2P network (no super-peers). The research shows that previous P2P, Grid, and Cycle Sharing systems tackled this problem by focusing on each resource type in isolation, such as (physical) resource discovery and service discovery. Methods to minimize storage and transmission costs were also researched. The current discovery mechanism's implementation only functions for static resources and was evaluated along side the Random Walk discovery method for comparison. The results were favorable over Random Walk, having higher query success rates with less hops while requiring a moderate increase in message size and storage space at each node (for routing information).

1 Introduction

There are millions of computers connected to the Internet[1] with more and more going online each day due to laptops, netbooks, PDAs, and smartphones. With so many devices connected to the same network, distributed computing on such a large scale cannot be ignored. As such, resource sharing has become immensely popular and has led to the development of Grid and Peer-to-Peer (P2P) infrastructures dedicated to that purpose. These infrastructures ease the sharing of various types of resources, that range from simple files, to software offering different services, and even hardware like CPUs and Printers.

The most popular form of resource sharing across the Internet is File Sharing via Peer-to-Peer applications, occupying roughly 50%-90% of all Internet traffic.[2] A lot of work has been done in this area to create robust and scalable systems, capable of efficiently supporting a large number of users in a decentralized manner. P2P Infrastructures can be divided between those that do not perform any node organization (Unstructured systems), such as Gnutella [1] and Freenet [2]; and those that structure their nodes to improve message routing (Structured systems), such as Chord [3], CAN [4], and Pastry [5].

* This work was supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds. Raoul Felix and this work were supported by FCT research project GINGER - PTDC/EIA/73240/2006

[1] http://www.internetworldstats.com

[2] http://torrentfreak.com/bittorrent-dominates-internet-traffic-070901

Grid and Cycle Sharing systems are similar in nature, as their objective is to perform large-scale parallel computations in scientific and commercial communities. While Grid systems harness the power of many interconnected networks of computers, which are usually centrally or hierarchically managed by the institutions that run them; Cycle Sharing systems take advantage of the many idle computers and game consoles already connected to the Internet, volunteered by home users.

Even though Peer-to-Peer and Grid systems are different, the literature [6–9] says that they will eventually converge. In this fashion, GINGER[3] [10], or simply GiGi, is a P2P Grid infrastructure that fuses three approaches (grid infrastructures, distributed cycle sharing, and decentralized P2P architectures) into one. GiGi's objective is to bring a Grid processing infrastructure to home users, i.e. a "grid-for-the-masses" (e.g. achieve faster video compression, face recognition in pictures/movies, high-res rendering, molecular modeling, chemical reaction simulation, etc.).

The common theme between these different systems is that users have a task that they want to accomplish: share files in P2P file sharing systems; perform scientific calculations in Grids; or perform CPU intensive tasks over a massive amount of idle home user computers in Cycle Sharing systems. Tasks require discoverable resources that satisfy certain requirements that can range from almost no requirements (file sharing), to simple requirements (idle CPU), to complex requirements (free CPU with X much RAM, with at least Y much storage space, and with application Z installed). This is where the work described in this paper comes in, where the objective is to create an effective, efficient, and scalable discovery protocol of resources, applications, and services for inclusion in the GINGER project.

The rest of this work is structured as follows. In Section 2 we discuss similar systems that also provide service or resource discovery. Section 3 describes the architecture of SERD[4], while in Section 5 we show some relevant performance results. Section 6 concludes this paper offering final remarks.

2 Related Work

This section can be divided into three main areas: i) efficient data representation where reducing the size of data storage and transmission is the objective, ii) resource discovery which only deals with the discovery of physical (e.g. CPU, RAM, etc.) or virtual (e.g. files) resources, and iii) service discovery where the main concern is discovering the services (e.g. facial recognition, high-resolution rendering, applications, etc.) provided by computers in a network.

2.1 Efficient Data Representation

Efficient Data Representation is important in this work because nodes have to store and transmit resource information about themselves and neighbors. **Compression** reduces the size of highly redundant information via a dictionary based (LZW [11]) or statistic based (Huffman coding [12]) encoding process. RSync [13] and the Low-Bandwidth File System [14] use **Chunks and Hashing** to divide data into chunks, calculating the hash of each chunk, and only transmitting those that have changed between versions of the same file. **Erasure codes** take another approach, and encode a message into a few symbols which can then be used to reconstruct a partially received message. Reperasure [15] uses this technique

[3] **G**rid **I**nfrastructure for **N**on-**G**rid **E**nvi**R**onments

[4] **S**calable and **E**fficient **R**esource **D**iscovery

to provide data replication without storing full-replicas. The three techniques, although important, are not directly applicable in this work. The reduced message size cancels the need to compress messages or divide them into chunks. We also do not need to perform any forward error correction nor replicate data.

The final and most useful technique is a space-efficient probabilistic data structure called **Bloom Filters**, which efficiently test whether an element is a member in a set with the possibility of a false-positive occurring. A set $S = \{x_1, x_2, ..., x_n\}$ of n elements is stored in an array of m bits all initially set to 0. It must also use k different hash functions, each of which maps some element to one position in the m bit array. Because Bloom filters are implemented as bit arrays, the union of two sets can be computed by performing the `OR` operation between the two, while their approximate intersections can be computed using the `AND` operation. Insertion is performed by passing the element through each of the k different hash functions and setting the resulting position in the m bit array to one. To test whether an element is in the set or not, it has to be passed through all hash functions and if all the resulting positions in the array are set to one, then the element has a high probability of being in the set. If any position has the value zero, then we know for definite that it is not in the set (no false negatives). The small false positive rate arises from the fact that when querying for an element that is not in the set, some hash functions may result in positions that were already used (have the value one) for a previously inserted item. Therefore, the more elements are inserted into the Bloom filter, the higher the chance of a query resulting in a false positive. Another shortcoming is the inability to remove an element from the Bloom filter, as simply setting the positions given by the k hash functions to zero have the side effect of removing other elements as well.

Bloom Filter variations exist to either extend their functionality or address some limitation. **Counting Bloom Filters** [16] allow both insertion and removal of elements by using an array of counters, instead of bits. In [17], Mitzenmacher shows that **Compressed Bloom Filters** can either occupy the same space but have a lower false-positive rate, or reduce their size and maintain their false-positive rate. Almeida et al. [18] created a **Scalable Bloom Filter** that dynamically grows in order to support the desired false-positive rate.

Finally, **Attenuated Bloom filters** were proposed in [19] to optimize search performance w.r.t. locality of objects. It uses an array of Bloom filters with depth d, where each row i, for $1 \leq i \leq d$, corresponds to the information stored at nodes i hops away. As the depth increases, more information will be stored in that Bloom filter row, making the respective filter more attenuated and resulting in a higher probability of false positives. Therefore, information closest to the node is more accurate, and less so the further away. The major advantage of this technique is that it permits us to efficiently locate objects up to d hops away, using as little storage space as possible (due to the Bloom filters) at the cost of a certain false positive rate. The disadvantage is that it *only* lets us search information about nodes up to d hops away.

2.2 Resource Discovery

Resource Discovery systems do a subset of what we want to accomplish with this work: locating physical or virtual resources to perform jobs. They can be split into three categories: Peer-to-Peer, Grid, and Cycle Sharing.

Peer-to-Peer systems do not distinguish between clients and servers; all nodes are equal and have no central coordination, making them decentralized. This leads to the various types of node topology organization: unstructured, structured, and hybrid. **Unstructured** system nodes are randomly connected to a fixed number of neighbors; there is no information about where resources

are located so message routing has to be performed by flooding. Searching can be uninformed or informed. Uninformed searches use no addition information to route queries, they are either flooded to all neighbors (Gnutella [1]), or are forwarded to a randomly selected neighbor (Iamnitchi et al. [20]). Informed searches are more intelligent and route messages based on collected information, but require more memory. Lie et al. [21] and the learning-based technique in Iamnitchi et al. forward queries to nodes that have replied to similar requests. Another strategy called best-neighbor in Iamnitchi et al. [20] just forwards queries to nodes with the highest success rate. **Structured** systems, such as Chord [3] and CAN [4] organize nodes into a rigid structure, called a Distributed Hash Table (DHT), which enables efficient exact-match query routing. Each node is assigned an identifier (key) which makes him responsible for all content (values) whose hash resolves to that key. Finally, **Hybrid** systems try to combine the best of both worlds without their disadvantages. Some systems in this category, like Pastry [5] and Kademlia [22], tend more towards structured systems, albeit with a less "rigid" structure, where any node belonging to a defined key subspace can act as a contact for those values. Others follow a more unstructured approach and use super-peers [23] that communicate between themselves on the behalf of less capable nodes (in terms of bandwidth or CPU performance), thus increasing routing performance.

Grid and **Cycle Sharing** systems share the same objective: to combine many geographically dispersed computer resources in order to perform tasks that require lots of CPU processing power, or that need to process huge amounts of data. Tasks like these are common when dealing with scientific, technical, or business problems. Grid systems can run in LAN environments such as that of a university, or in a much larger network compromised of interconnected networks that belong to different institutions, corporations, or universities. Condor [24] and Legion [25] are typical examples of such systems, where information about all resources are stored in a central component, known as the Matchmaker in Condor, and in Legion is divided into 3 subcomponents: the Collection, Scheduler, and Enactor. This central component receives job requests, tries to match their requirements to available resources, and reserve those resources while notifying the requester. Cycle Sharing systems rather operate over the Internet, which can be highly unreliable with variable connection quality. Another important difference is that anyone with a computer can join a cycle sharing project of interest (e.g. SETI@Home [26] or Folding@Home) and volunteer their resources during idle times. This brings the additional problem of unreliable peer connections and possibly forged results from untrusted peers.

2.3 Service Discovery

Service Discovery systems, like Resource Discovery, do the missing subset of this work: enabling the automatic detection of services provided by computers in small LAN environments, like home networks, or in large-scale enterprise networks, like a corporation or university. SLP [27] and Jini [28] use a client/server architecture, where servers collect service information and perform lookups for clients. SLP can function without directory servers using multicast to find services, but only in small LAN environments.

The systems presented by Goering et al. [29] and Lv and Cao [30] use a Peer-to-Peer architecture instead, with the objective of being able to function in ad-hoc networks. Goering et al. propose a service discovery protocol based on the use of Attenuated Bloom Filters, which provide a method to locate objects, giving preference to objects located nearby. It is simply an array of Bloom Filters of depth d, where each row represents objects at different distances which, in this

case, is in term of hops. Each node has an Attenuated Bloom Filter for each of its neighbors, which is consulted when a query is received in order to send it in a direction it will have a higher chance of success. The first level of the Attenuated Bloom Filter corresponds to the services that are one hop away, the second to services two hops away, and so forth. Therefore, the larger the distance from the node, the more services will be contained in the corresponding Attenuated Bloom Filter which will increase the chance of false positives. Relying solely on Attenuated Bloom Filters gives this system a big limitation: only the services located up to d-hops away can be easily found. Lv and Cao resolve this drawback by having nodes more than $d+1$ hops away cooperate among themselves. Thus, when a query is received, it follows the same process of checking the Attenuated Bloom Filters of its neighbors like Goering et al, but if no services are found, then the query is forwarded to a node $d+1$ hops away where the search begins again.

3 Architecture

The objective of this work is to enhance the resource discovery mechanism in GINGER [10], also known as GiGi, by making it completely decentralized and more complete. This completeness regards the system's ability to discover, not only basic resources (e.g. CPU, Bandwidth, Memory, etc.), but also specific installed applications (e.g. video encoders, simulators, etc.) and services (e.g. face recognition, high-res rendering, etc.). Because GiGi can be used in many different ways ("grid-for-the-masses"), it has to be flexible enough to run different types of jobs normally performed by home-users.

In order to cope with a dynamic peer population and high churn rate, this system uses an unstructured peer-to-peer approach to resource discovery, even though message routing may not have optimum efficiency. If a structured system were to be used, the messages needed to keep the structure intact with an unstable population, such as home-users, could possibly result in a high overhead. Attenuated Bloom Filters are used to enhance message routing and speed up resource location. Note that this solution is different to the systems mentioned in the Related Work because it combines all types of different resources into one discovery mechanism. It is especially different to the works [29,30] that also make use of Attenuated Bloom Filters due to usage of one aggregated Attenuated Bloom Filter (explained next), and the fact that all the different types of basic resources, services, and applications are encoded in the Bloom Filter.

Each node in the network stores a cached version of the Attenuated Bloom Filters of their neighbors. This information is then merged into one single Attenuated Bloom Filter by inserting the union (`OR` operation) of all neighbor Bloom Filters at a certain depth k into depth $k+1$ (Figure 1). The consequence of using an Attenuated Bloom Filter of, for example, depth $d = 2$ is that a node will only know about the resources of nodes up to 2 hops away. A solution for this problem is discussed further in Section 3.

Discovery Mechanism: The discovery of resources, applications, and services (illustrated as a flowchart in Figure 2) will be performed in the following way. When a node receives a query, it will check its own information to see if it can satisfy the requirements. If it does, a reply is sent directly to the node that originated the query. If not, it goes through its aggregated Attenuated Bloom Filter, which contains the combined information from its neighbors Attenuated Bloom Filters. This way, we can quickly determine if the query cannot be satisfied with nodes up to d hops away, in which case it will be sent directly to a node $d+1$

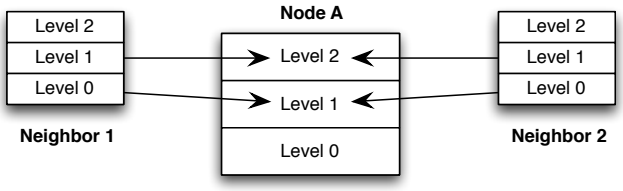


Fig. 1. Example of a node A creating a single Attenuated Bloom Filter by merging each Level i of its neighbors' Attenuated Bloom Filters into Level $i + 1$.

hops away to restart the search. If the query can be satisfied with nodes at most d hops away, the node then needs to determine the direction to send the query for it to be resolved. This is done by checking all the cached Attenuated Bloom Filters of its neighbors to determine which one has the requested resources. If found, it then forwards the query to that neighbor. If not, then it is because the aggregated Attenuated Bloom Filter returned a false positive, which is mitigated by simply sending the query to a node more than $d + 1$ hops away so it can be resolved. As each message is forwarded to a node, the sender adds his own ID to the resource query's Bloom Filter which keeps track of where the message has been sent. This Bloom Filter is cleared when a query jumps to a node $d+1$ hops away. If any node received a query message and its ID is in the Bloom Filter, then there must have been a false positive and therefore the query should fail.

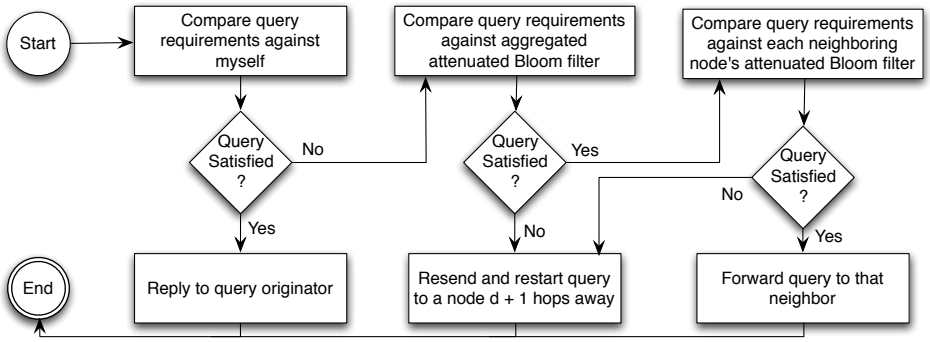


Fig. 2. Flowchart of resource, service, and application discovery from Section 3

Dynamic Resources: Some resources are mostly static and do not change, like the Operating System, CPU and Disk speed, certain application versions, etc. But there are other resources whose values can change quite often, such as amount of RAM occupied, amount of CPU in use, etc. For those cases, if we used a classic Bloom Filter then it would need to be rebuilt periodically since it does not support the removal of elements. More, this rebuilding procedure would require sending information about resources that are not expected to change, thus wasting bandwidth.

Therefore, instead of using a classic Bloom Filter to store the information about the dynamic resources, a Counting Bloom Filter is used. To compensate the fact that a Counting Bloom Filter occupies more bits than a classic one, we use a smaller size, as the number of static resources is greater than dynamic ones. The usage of this new Bloom Filter mirrors that described in the previous sections: queries for dynamic resources use the Aggregated Counting Bloom Filters

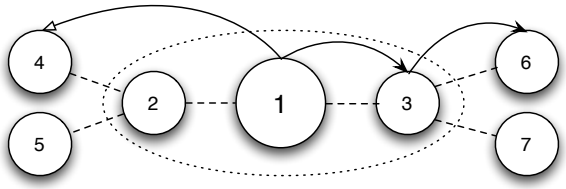


Fig. 3. Example showing how resource queries are forwarded with an Attenuated Bloom Filter of $d = 1$. When a neighbor has information about the desired resource, such as Node 3, then query is forwarded to that peer, who in turn forwards the query to Node 6 which contains the resource. In another case, when there is no information about the desired resource in Node 1's area (consisting of Nodes 1, 2 and 3), then the query is forwarded to an Outer Limit Node 4, where the search is then restarted.

instead. The difference now is when a dynamic resource changes its value and passes a certain threshold, the direct neighbors of that node are notified. Thus, the information closest to the node with the resource is kept up to date. The updating of nodes further away occurs at a later stage, when there are enough resource value alterations that can be sent in a batch, in order to save messages.

Outer Limit Peer Discovery: Using an Attenuated Bloom Filter of a certain depth d limits the amount of information a node has about its surrounding neighbors. If a query is received and cannot be satisfied using the information the node knows about its peers in the same area, then it forwards the query to another node that is $d + 1$ hops away (which, conceptually, is part of another area).

To find those Outer Limit Peers, the system uses a simple Random Walk strategy to forward a discovery message until it reaches a node $d + 1$ hops away. Once that node is found, it replies to the message originator. The Peer Discovery protocol has two parameters which can be fine tuned, such as: width (w) and length (l). Width represents the number of nodes the discovery query should be sent to in parallel, and the length is the number of hops that the outer limit node should have. On the off chance that a node does not know about any outer limit peers, either due to particular topology configurations or node failures, the system just forwards the query to a random neighbor.

3.1 Resource Representation

Information about resources, applications, and services that each node offers are represented inside a Bloom Filter. But, because a Bloom Filter is only capable of performing membership tests given a key (in this case a string), we need to add information about the actual resource (like type, value, etc.) to that key on insertion for it to be useful in discovering resources. Therefore, keys use namespaces to differentiate between resources and their values, which also helps with performing membership tests for resources. The naming convention uses a 3-level namespace, each separated using the colon (":") as a delimiter, and follows the following rules:

- *Level 1*: Name of the Resource, Service, or Application (e.g. CPU or ffmpeg)
- *Level 2*: Type of the Resource, Service, or Application (e.g. MHz or version)
- *Level 3*: Actual value of the Resource, Service, or Application

For instance, if we wanted to store the fact that a node has a CPU of 3 GHz, the key we would insert into the Bloom Filter would be: "CPU:GHz:3". Or, if a node has the application ffmpeg version 2.3 installed, the key would look like: "ffmpeg:version:2.3". But, for different nodes to be able to communicate with each other and search for the same resources, the naming of resources, services, and applications need to be the same between all of them. An ontology could be used, but that is out of the scope of this work. For the time being, the system reads a configuration file that specifies the name of the resource among other things. This configuration file needs to be the same for all nodes in the network.

Insertion: However, just following a naming convention will not suffice for the discovery of resources. We also need to take into account the values used for each resource. If we do not restrict the possible values, we would need to employ a brute force strategy when querying for resources, trying each value combination and testing the Bloom Filter. For example, to find a node that at least contains a CPU of 2.6 GHz, we would need to test for values such as 2.6, 2.7, 2.8, 2.9, 3.0, etc., which is highly inefficient. To speed this up, we define a *minimum*, *maximum*, and a *quantum* for each resource value type (which are also specified in a configuration file). The *minimum* (resp. *maximum*) is the smallest (resp. largest) value that the resource will have encoded in the Bloom Filter. The *quantum* defines how the value space, from *minimum* to *maximum*, will be divided. When a resource is inserted into the Bloom Filter, it is first inserted with the key that corresponds to its range, and then with all the other keys that correspond to ranges smaller than the resource's value. For example, if we define $minimum = 0$, $maximum = 4000$, and $quantum = 1000$ for CPU values in MHz, then the range of values is divided into the following segments: $]0, 1000]$; $]1000, 2000]$; $]2000, 3000]$; and $]3000, 4000]$. Or, if a CPU of 999MHz were to be inserted into the Bloom Filter, it would need to be inserted under the value 1000: "CPU:MHz:1000"; and so on.

Querying: Now, when querying a Bloom Filter for a value, the range the value falls under needs to be determined for the specified resource and checked. For instance, if a query requires a CPU of at least 2600 MHz, we would only need to perform one exact match query using the range the value in the requirements belongs to, which in this case is 3000 ($2600 \subset]2000, 3000]$). Therefore, we only need to test the key "CPU:MHz:3000" against a Bloom Filter because processors with a faster CPU will also be registered under this key. This strategy avoids the brute-force approach and efficiently speeds up the querying process. However, one needs to take care when specifying the *quantum* value due to precision problems. In this example, a CPU of at least 2600 MHz is required, but testing the Bloom Filter with key "CPU:MHz:3000" can result in CPUs that belong to the interval $]2000, 2599]$, thus not satisfying the requirements. In a real-world system, using a $quantum = 200$ would probably be more suitable, giving enough precision without requiring too much overhead. This, and searching for a resource with a key one *quantum* value higher than required will ensure query satisfaction.

4 Implementation Details

This work is implemented using the PeerSim [31] simulator with its Event Driven capabilities, approximating the simulation more to real-life as opposed to a Cycle Driven simulation. Because PeerSim is implemented in Java, the SERD discovery mechanism is also implemented in Java, which also allowed us to use an open source Bloom Filter implementation from the well known Hadoop project, providing us a certain amount of confidence w.r.t. its quality.

In order to be able to evaluate this work, we had to build an infra-structure around PeerSim to allow things such as topology creation, resource distribution, and node activity specification. The **topology** of a network can either be loaded using a file that describes the connections between nodes, or can be generated randomly using parameters that ensure minimum and maximum number of neighbors. **Resource distribution** among nodes can be performed in a static way using a simple file that specifies which node should have what resource; or, it can be specified in a more random fashion by specifying criteria to select a certain number of nodes. Distribution criteria can be the number of hops between nodes, the density/frequency of nodes that have the resource, or even the homogeneity of resource distribution. **Node activity specification** also uses a text file where nodes can be selected using various types of specifiers (e.g. randomly, exact match, nodes with a certain resource, etc.) along with the actions that they should perform (e.g. search for some resource) and when that action should be executed (in terms of simulation cycles or periodicity).

Another implementation issue we had was the initial construction of the Attenutated Bloom Filters. PeerSim starts with an already defined topology, so we simulated a joining phase on top of PeerSim for nodes to exchange resource information when a new peer enters the network.

5 Evaluation

PeerSim was used to evaluate SERD in a virtual network environment with six different test scenarios. These include varying the number of nodes that have the desired resource, which ranges from *very abundant* (50% of the nodes have the resource), *abundant* (25% have the resource), and *scarce* (only 5% have the resource). For each of those cases, we also vary the values of the resources, separating them into two groups: *uniform*, which is common with an application like GCC, either a computer has it or it does not; and *non-uniform* where values vary quite a bit, similar to a Hard Drive where the range we used was 0GB to 1000GB.

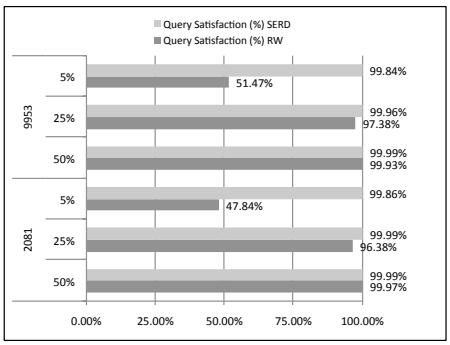


Fig. 4. Query Satisfaction for GCC (*uniform*)

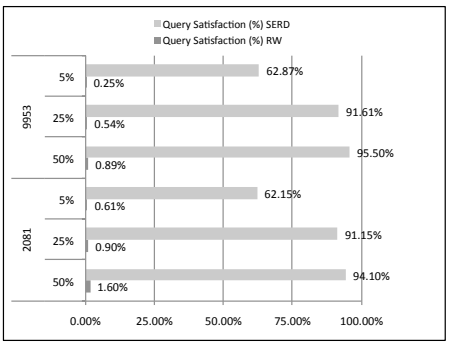


Fig. 5. Query Satisfaction for Hard Drive (*non-uniform*)

The SERD protocol was compared against the Random Walk protocol, which is used as a touchstone as it is easy to implement and functions as a baseline for performance (no protocol should perform worse). The RW implementation uses exact-match searches and just forwards queries to random neighbors.

Each scenario was tested with both RW and SERD protocols using two different network sizes: one with 2081 nodes and another with 9953 nodes, repre-

senting small and large networks respectively. Neighbors in this topology were randomly assigned, with the maximum number of neighbors being three. 10% of the nodes were randomly chosen to periodically send a query, in parallel, for a certain resource based on the scenario. Query messages were sent with a $TTL = 2 * log_2(NETWORK\_SIZE)$ to make sure resource queries eventually fail. As SERD uses an Attenuated Bloom Filter, the chosen depth for the test was $d = 3$.

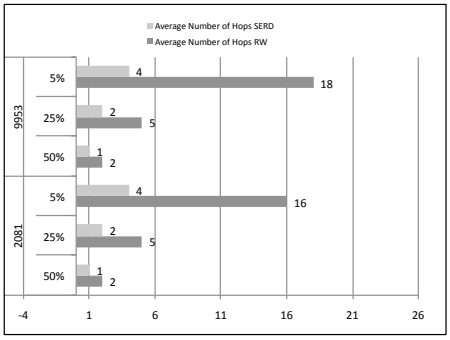


Fig. 6. Average Number of Hops for GCC (*uniform*)

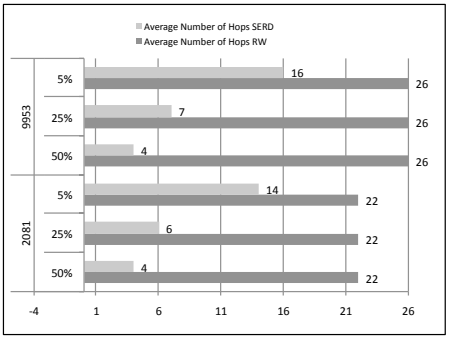


Fig. 7. Average Number of Hops for Hard Drive (*non-uniform*)

Figures 4 and 5 show the percentage of resource queries that were satisfied (out of 7072 and 33830 sent queries for network sizes of 2081 and 9953, respectively). SERD proved to be able to find the requested resources with a percentage of satisfaction consistently superior than 90%, with the exception of the *scarce* scenarios with *non-uniform* values. Still, more than half of the resource queries were satisfied even though the resources were distributed to only 5% of the nodes, further complicating the search. RW did well in the *uniform* scenario, but struggled in the *scarce* one. RW performed terribly with *non-uniform* values, barely being able to satisfy any queries. This happened because the protocol used exact-match and just randomly picked a neighbor to forward a message to, which led to dead ends.

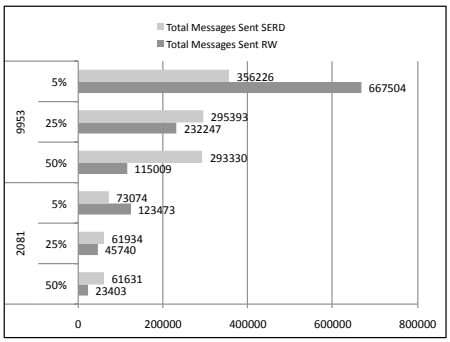


Fig. 8. Total Messages Sent for GCC (*uniform*)

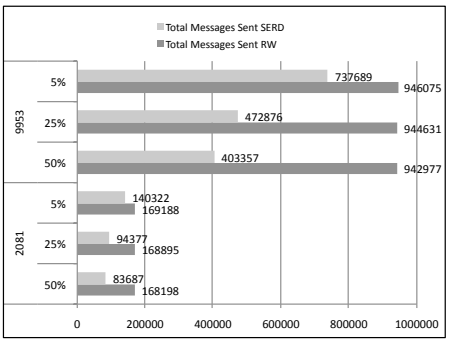


Fig. 9. Total Messages Sent for Hard Drive (*non-uniform*)

With regards to the average number of hops the resource queries traveled, seen in Figures 6 and 7, the scenarios that proved tougher had messages travel

a lot more. When searching for GCC, both RW and SERD protocols had a low number of hops, except for the *scarce* scenario were RW increased quite a bit. While looking for a Hard Drive, the average number of hops for RW were close or equal to the TTL (22 and 26 for network sizes of 2081 and 9953, resp.), which is obvious seeing as almost all queries failed.

In Figures 8 and 9 we can see the total number of sent message. With the exception of the RW protocol's search for Hard Drives, the SERD protocol uses more messages than the RW protocol. This is to be expected because SERD needs to exchange neighbor resource information (the Attenuated Bloom Filters) and look for Outer Limit Peers, unlike RW. As this is a work in progress, this is one area that we will try to optimize in order to reduce the number of messages.

Finally, the average message size and average routing information storage size occupied by each node can be seen in Figure 10. SERD messages are almost double the size of RW messages, which is expected because SERD messages include a Bloom Filter to keep track of nodes the message has passed through. With regards to the information stored at each node, SERD uses much more space than RW because RW nodes only keep information about their own resources, whereas SERD nodes store the Attenuated Bloom Filters of its neighbors and needs space for its own Aggregated Attenuated Bloom Filter.

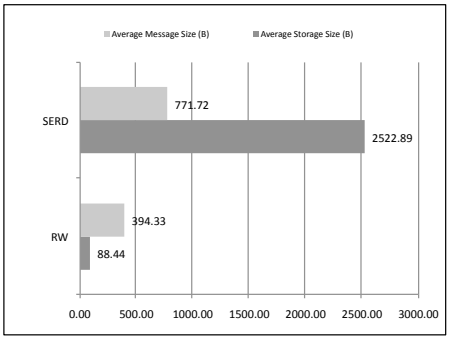


Fig. 10. Average Message Size and Average Storage Size at each Node

6 Conclusion

GiGi [10] allows home users to take advantage of Grid computing which was previously only available to scientific and corporate communities. Tasks that would usually take a lot of time, such as audio and video compression, signal processing related to multimedia content (e.g. photo, video, and audio enhancement), intensive calculus for content generation (e.g. ray-tracing, fractal generation), among others, can now be sped up by parallelizing and distributing them over many computers.

However, to distribute the tasks GiGi needs to locate the resources that satisfy task prerequisites. This is precisely what the architecture described in this paper does: discovering physical resources, services, and applications of computers connected to the same P2P Grid. The main objectives are to create a decentralized discovery mechanism that is efficient and scalable for the GiGi project. Even though this work is for the GiGi project, it is completely independent and can be used in other types of networks, such as cycle-sharing networks.

The current implementation focuses mainly on static resources (work in progress) and was evaluated alongside another, albeit simpler, discovery mechanism called Random Walk (RW). Results show that SERD proved to be better than RW,

with higher query success rates using less hops at the expense of increased message size and storage space. There is still work to be done to increase the efficiency and scalability of the system.

References

1. Gnutella Protocol Specification. Last checked: 2009-12-18. http://wiki.limewire.org/index.php?title=GDF.
2. I. Clarke, S.G. Miller, T.W. Hong, O. Sandberg, and B. Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.
3. I Stoica, R Morris, D Karger, and M Kaashoek. Chord: A scalable peer-to-peer lookup service for internet applications. *Proceedings of the 2001 conference on Applications*, Jan 2001.
4. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, page 172. ACM, 2001.
5. A Rowstron and P Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Lecture notes in computer science*, pages 329–350, Jan 2001.
6. S Androutsellis-Theotokis and D Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, Jan 2004.
7. I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. *Lecture Notes in Computer Science*, pages 118–128, 2003.
8. D. Talia and P. Trunfio. Toward a synergy between p2p and grids. *IEEE Internet Computing*, 7:96–96, 2003.
9. A. Iamnitchi and D. Talia. P2p computing and interaction with grids. *Future Generation Computer Systems*, 21(3):331–332, 2005.
10. L Veiga, R Rodrigues, and P Ferreira. Gigi: An ocean of gridlets on a" grid-for-the-masses. *Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007. CCGRID 2007*, pages 783–788, 2007.
11. M Nelson. Lzw data compression. *Dr. Dobb's Journal*, Jan 1989.
12. D Huffman. A method for the construction of minimum-redundancy codes. *Resonance*, Jan 2006.
13. A. Tridgell. Efficient algorithms for sorting and synchronization. *Doktorarbeit, Australian National University*, 1999.
14. A Muthitacharoen, B Chen, and D Mazieres. A low-bandwidth network file system. *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 174–187, Jan 2001.
15. Z Zhang and Q Lian. Reperasure: Replication protocol using erasure-code in peer-to-peer storage network. *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, pages 330–339, Jan 2002.
16. Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.
17. Michael Mitzenmacher. Compressed bloom filters. *IEEE/ACM Trans. Netw.*, 10(5):604–612, 2002.
18. PS Almeida, C Baquero, N Preguiça, and D Hutchison. Scalable bloom filters. *Information Processing Letters*, 101(6):255–261, 2007.
19. Sean C Rhea and John Kubiatowicz. Probabilistic location and routing. 2002.
20. A Iamnitchi, I Foster, and D Nurmi. A peer-to-peer approach to resource location in grid environments. *INTERNATIONAL SERIES IN OPERATIONS RESEARCH AND MANAGEMENT SCIENCE*, pages 413–430, Jan 2003.
21. L Liu, N Antonopoulos, and S Mackin. Social peer-to-peer for resource discovery. *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 459–466, Jan 2007.
22. P Maymounkov and D Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. *Proceedings of IPTPS02*, Jan 2002.
23. C Mastroianni, D Talia, and O Verta. A super-peer model for building resource discovery services in grids: Design and simulation analysis. *Lecture notes in computer science*, 3470:132, Jan 2005.
24. D Thain, T Tannenbaum, and M Livny. Condor and the grid. *Grid Computing: Making the Global Infrastructure a Reality*, pages 299–335, Jan 2003.
25. S Chapin, D Katramatos, and J Karpovich. Resource management in legion. *Future Generation Computer Systems*, Jan 1999.
26. D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):61, 2002.
27. E Guttman. Service location protocol: Automatic discovery of ip network services. *IEEE Internet Computing*, Jan 1999.
28. J Waldo. The jini architecture for network-centric computing. *Communications of the ACM*, Jan 1999.
29. P Goering and G Heijenk. Service discovery using bloom filters. *Proc. Twelfth Annual Conference of the Advanced School for Computing and Imaging, Belgium*, Jan 2006.
30. Qingcong Lv and Qiying Cao. Service discovery using hybrid bloom filters in ad-hoc networks. *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, pages 1542–1545, 2007.
31. PeerSim. Last checked: 2009-12-27. http://peersim.sourceforge.net/.

Thicket: Construção e Manutenção de Múltiplas Árvores numa Rede entre Pares*

Mário Ferreira, João Leitão, and Luís Rodrigues

INESC-ID / IST
{mvvf, jleitao}@gsd.inesc-id.pt, ler@ist.utl.pt

Resumo As árvores de disseminação permitem a distribuição eficiente de conteúdos em redes sobrepostas mas impõem uma carga aos nós interiores muito superior à dos nós folha. Uma forma de contornar este problema passa pela utilização de múltiplas árvores em que cada nó é apenas interior num pequeno subconjunto de todas as árvores, e folha nas restantes. As múltiplas árvores permitem a distribuição da carga e o envio de informação redundante para a recuperação de falhas. Neste trabalho propomos o Thicket, um algoritmo para a construção e manutenção de múltiplas árvores, de forma totalmente descentralizada, sobre uma rede não estruturada. O algoritmo foi implementado e avaliado através de simulações, utilizando uma rede composta por 10.000 nós.

Abstract Spanning tree structures allow efficient resource usage when broadcasting data on overlays networks but they impose a much higher load to the interior nodes than the leafs. One way of overcoming this issue is to employ multiple spanning trees where a node is interior in just a few of them and a leaf in the remaining. This configuration enhances load distribution and provides a mechanism for introducing redundancy required for fault-tolerance. This work presents Thicket, a protocol for building and maintaining multiple spanning trees, with fully decentralized algorithms, in an unstructured overlay. The protocol was implemented and evaluated, using simulations, in a network composed of 10.000 nodes.

1 Introdução

Os mecanismos que permitem a distribuição de informação de forma fiável e eficiente, para um conjunto considerável de participantes, são extremamente úteis para um grande número de aplicações, desde sistemas de controlo e monitorização[1], até transmissão de vídeo ao vivo e serviços de Televisão sobre IP (IPTV)[2]. Neste trabalho abordamos mecanismos de disseminação entre-pares baseado na cooperação dos seus participantes.

O principal problema que afecta este tipo de sistemas é o desbalanceamento na contribuição de cada nó na distribuição de conteúdos. Utilizando uma árvore de disseminação é possível distribuir a carga de reencaminhamento pelos nós interiores, porém, os nós folha apenas recebem dados não contribuindo para a disseminação. Note que, existe uma fracção substancial de nós folha numa rede deste tipo.

O nossa solução utiliza a abordagem baseada na construção de múltiplas árvores de disseminação sobre uma rede não estruturada, promovendo a utilização eficiente dos recursos disponíveis e evitando redundância desnecessária

* Este trabalho foi parcialmente suportado pelo financiamento pluri-anual do INESC-ID através do programa PIDDAC e pelos projectos "Redico" (PTDC/EIA/71752/2006).

que advém da utilização de inundação ou de difusão epidémica. Porém, a utilização de uma árvore impõe uma carga muito superior aos nós interiores do que aos nós folha. Para além disso, a falha de um nó interior provoca quebras na árvore afectando a fiabilidade da disseminação. Uma forma de contornar estes problemas consiste em utilizar múltiplas árvores nas quais cada nó é interior em apenas uma ou num número reduzido de árvores, e folha nas restantes. As múltiplas árvores permitem a distribuição da carga do sistema por todos os nós e, também, o envio de dados redundantes por diferentes árvores de forma a tolerar falhas de nós ou ligações.

Neste trabalho apresentamos o Thicket, um algoritmo para construir e manter múltiplas árvores de forma descentralizada numa rede sobreposta. Este algoritmo aborda uma região pouco explorada do espaço de soluções. As redes não estruturadas são mais flexíveis e acomodam mudanças na configuração do sistema mais facilmente que as redes estruturadas, como é o caso das Tabelas de Dispersão Distribuídas (*Distributed Hash Tables*, DHTs), pois não impõe uma topologia específica na rede.

Este artigo encontra-se organizado da seguinte forma. A Secção 2 motiva este trabalho analisando as soluções existentes e discutindo as suas vantagens e limitações. De forma a evidenciar os desafios da nossa abordagem, na Secção 3, demonstramos as limitações de algumas soluções simplistas para o problema. A apresentação e descrição do protocolo é feita em detalhe na secção 4, sendo os resultados experimentais apresentados na Secção 5. Finalmente, a Secção 6 conclui o artigo.

2 Trabalho Relacionado

Existem essencialmente, três abordagens para a distribuição de informação em grande escala em sistemas entre-pares: a *difusão epidémica*, a abordagem em *árvore*, e a abordagem em *árvore embebida*. Na difusão epidémica[3,4] a fonte envia a mensagem para f[1] nós escolhidos de forma aleatória. Quando um nó recebe uma mensagem pela primeira vez, o processo é repetido. Esta abordagem é simples, escalável e robusta. Infelizmente, não utiliza os recursos eficientemente, pois a sua robustez é obtida à custa de aumento significativo da redundância dos dados. A abordagem em árvore consiste em organizar os intervenientes de forma a obter uma rede sobreposta em que a topologia corresponde a uma árvore tolerante a falhas [5]. A principal vantagem de utilizar uma árvore é o aproveitamento eficiente dos recursos. Por outro lado, uma árvore é difícil de manter em ambientes instáveis. Desta forma, esta solução não é adequada para sistemas de grande escala sujeitos a alterações constantes da sua filiação. Finalmente, a abordagem em árvore embebida consiste em usar mecanismos eficientes para construir uma árvore sobre uma rede sobreposta já existente[6,7].

As abordagens em árvore embebida podem ser aplicadas em redes estruturadas[7] ou não estruturadas [6,8]. Soluções baseadas em redes não estruturadas são potencialmente mais resistentes à variação da filiação do sistema, dado que impõem menos restrições à topologia da rede e podem ser rapidamente reparadas.

Por outro lado, as soluções baseadas em árvore podem ser também divididas em soluções de árvore única ou com árvores múltiplas. As soluções com uma única árvore são mais simples mas possuem dois problemas: utilizam os recursos

[1] f é um parâmetro típico destes sistemas designado por *fanout*.

do sistema de forma desbalanceada (nós interiores gastam mais recursos para enviar dados aos seus filhos enquanto que nós folha apenas recebem dados) e são mais susceptíveis a quebras devido à falha de nós interiores da árvore. As soluções de árvores múltiplas constroem várias árvores ligando os mesmos participantes. As árvores são construídas de forma que cada nó seja interior numa ou num número reduzido das árvores existentes e folha nas restantes. Esta abordagem promove o balanceamento da carga do sistema, pois todos os nós reenviam dados. Para além disso, enviando informação redundante em algumas árvores (por exemplo utilizando técnicas de *network coding*[9]), é possível tolerar faltas visto que a falha de um nó apenas quebra a árvore onde este é interior, os receptores continuam a conseguir obter os dados a partir das restantes árvores.

Estas últimas abordagens podem ainda ser classificadas segundo o tipo de algoritmo utilizado na construção das árvores. Algoritmos centralizados dependem de nós específicos, que contêm informação global acerca da topologia do sistema. É de notar que, mesmo um algoritmo centralizado não é trivial, dado que o problema da construção óptima das múltiplas árvores é NP-completo[10]. Estas soluções são, contudo, pouco interessantes para sistemas de grande dimensão, pois não possuem capacidade de escala nem tolerância a faltas. Alternativamente a comunidade tem proposto soluções descentralizadas. Exemplos de algoritmos descentralizados são o SplitStream[11] e o Chunkyspread[12].

O SplitStream baseia-se numa variante do Scribe para construir várias árvores de disseminação disjuntas sobre a DHT do Pastry[13]. Tal como na nossa abordagem, os autores tentam construir árvores em que um nó é interior em apenas uma árvore e controlam o número de filhos de um nó na árvore em que este é interior (*i.e.*, limitando a carga de cada nó) de acordo com a sua capacidade. Porém, os autores utilizam uma DHT; os nós são interiores numa única árvore por desenho, dado que cada árvore possui uma fonte cujo identificador tem um prefixo distinto. Note-se que manter uma DHT tem um custo muito superior comparado com uma rede não estruturada. Para além disso, a rede não estruturada pode recuperar mais rapidamente que o Pastry: no Pastry um nó que falha apenas pode ser substituído por nós cujo identificador é adequado para a posição em causa (de acordo com a lógica da rede estruturada). Adicionalmente, o esquema utilizado para assegurar o grau máximo dos nós interiores pode resultar na desconexão de vários nós da árvore prejudicando a fiabilidade do protocolo. O SplitStream também utiliza ligações adicionais para além das oferecidas pelo Pastry, que acarretam custos de manutenção mais elevados.

O Chunkyspread[12] é um protocolo que constrói e mantém várias árvores de disseminação sobre uma rede sobreposta não estruturada. Este protocolo limita ainda a carga e grau dos nós de acordo com a sua capacidade. Contudo, o mecanismo utilizado não controla o número de árvores em que um nó é interior. Esta lacuna permite a criação de árvores dependentes entre si, *i.e.*, onde um nó é interior em várias árvores. Esta propriedade é indesejável do ponto de vista da fiabilidade. Na Secção 5, demonstramos que, em cenários onde ocorrem falhas de nós, é de extrema importância que as árvores construídas sejam independentes.

Em resumo, o nosso objectivo passa por desenhar uma solução que combine as seguintes funcionalidades: i) cria um árvore embebida numa rede sobreposta, oferecendo eficiência e robustez; ii) opera de forma completamente descentralizada; iii) constrói múltiplas árvores com poucos nós interiores em comum; e iv) operam sobre redes não estruturadas. A Tabela 1 ilustra as várias combinações

	Parcialmente Centralizado	Descentralizado	
		Rede Estruturada	Rede Não Estruturada
Single tree	Bayeux[14]	Scribe[7]	MON[1], Plumtree[6]
Multiple tree	CoopNet[15]	Splitstream[11]	Chunkyspread[12], **Thicket**

Tabela 1. O Thicket no espaço de soluções

no espaço de desenho para o problema, identificando as soluções existentes em cada região e localizando a nossa solução nesse espaço.

3 Algumas Abordagens Simplistas

Como referido anteriormente, o nosso objectivo é desenhar um algoritmo descentralizado para construir t árvores sobre uma rede não estruturada. À primeira vista este objectivo pode parecer simples. Em particular, poderíamos considerar um algoritmo que estenda de forma trivial o trabalho já existente. Duas alternativas surgem como candidatas:

O SplitStream[11] constrói múltiplas árvores sobre uma rede estruturada. Pode-se tentar usar uma abordagem semelhante sobre uma rede não estruturada. Em particular, podemos escolher t nós aleatoriamente, e construir uma árvore distinta com raiz em cada um desses nós. Esta abordagem é uma versão simplificada do protocolo Chunkyspread. Denominámos esta abordagem de *Naive Unstructured spliTStream*, ou simplesmente, NUTS.

O Plumtree[6] constrói uma única árvore, de forma descentralizada, sobre uma rede não estruturada. Desta forma, é possível considerar a solução simples que consiste em executar este algoritmo t vezes, embebendo t redes sobrepostas distintas e criando uma árvore em cada uma delas. A intuição desta abordagem é que a aleatoriedade do processo de construção das redes sobrepostas (e das respectivas árvores) é suficiente para criar árvores diversificadas. Denominámos esta abordagem de *Basic multiple OverLay-TreeS*, ou simplesmente, BOLTS.

Implementámos estas duas estratégias simplistas para verificar o seu desempenho. Analisamos os resultados de forma a tirar conclusões para o desenho da nossa solução. Na realização destas experiências utilizámos o HyParView[16] para construir a rede sobreposta. A topologia criada pelo HyParView é semelhante a um grafo regular aleatório o que facilita o balanceamento da carga. Para construir as árvores utilizámos o protocolo Plumtree[6]. De forma a experimentar a abordagem NUTS, construímos uma rede HyParView e utilizámos o protocolo Plumtree para criar t árvores com raiz em nós escolhidos aleatoriamente. Para experimentar a estratégia BOLTS, criamos t instâncias independentes de redes HyParView e, de seguida, criarmos uma árvore em cada uma destas instâncias.

Avaliámos ambas as estratégias simulando um sistema composto por 10.000 nós e construindo 5 árvores de disseminação. Para a abordagem NUTS utilizámos uma única rede sobreposta com grau de 25. No caso do BOLTS configurámos cada instância HyParView para ter grau 5. Estas configurações asseguram que ambas as abordagens contêm o mesmo número de ligações, aproximadamente.

A Figura 1 mostra a percentagem de nós interiores em 0, 1, 2, 3, 4 e 5 árvores. Em ambas as estratégias apenas uma pequena fracção dos nós (entre 7% e 17%) são interiores numa única árvore. A maioria dos nós do sistema são interiores em

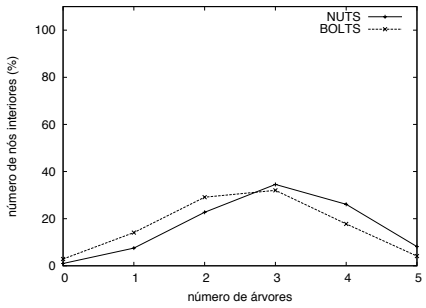


Figura 1. Distribuição de nós k-interiores.

2, 3 ou 4 árvores (com uma pequena percentagem de nós em todas as árvores). Estas estratégias criam configurações sub-óptimas, onde muitos nós reenviam mensagens em mais do que uma árvore. Adicionalmente, a falha de um único nó pode quebrar várias, ou mesmo todas as árvores, o que compromete claramente a fiabilidade do sistema.

4 Thicket

O Thicket foi concebido para operar sobre uma rede não estruturada, que implementa um protocolo de vizinhança reactivo e exporta uma vista parcial simétrica do sistema[2]. Este protocolo é responsável por notificar a camada do Thicket quando ocorrem alterações na vista parcial de um nó utilizando as funções *NeighborUp*(p) e *NeighborDown*(p). O Thicket utiliza uma técnica baseada em difusão epidémica para construir T árvores divergentes, de forma a que a maioria dos nós seja interior em apenas uma árvore e folha nas restantes. As restantes ligações da rede sobreposta são usadas para: i) assegurar a cobertura das árvores sobre todos os nós; ii) detectar e recuperar de situações de falha de nós onde ocorrem partições de uma ou mais árvores; iii) assegurar que a altura das árvores se mantém num valor baixo, mesmo na presença de falhas; e, finalmente, iv) assegurar que a carga imposta pelo reenvio de dados a cada participante é limitada por um parâmetro *maxLoad*.

Restrições de espaço obrigam-nos a descrever o funcionamento do algoritmo de forma sumária, referindo apenas os principais mecanismos subjacentes à sua operação. Uma descrição pormenorizada, incluindo pseudo-código que captura a operação de cada mecanismo pode ser encontrado em [17].

Cada nó n mantém um conjunto $backupPeers_n$, que contém os identificadores dos vizinhos que não são utilizados para receber (ou reencaminhar) mensagens em nenhuma das T árvores. Inicialmente, todos os vizinhos de n estão neste conjunto. Para cada árvore t mantida pelo Thicket, cada nó armazena um conjunto $t.activePeers_n$ com os identificadores dos vizinhos utilizados para receber (ou reencaminhar) mensagens de dados em t. Cada nó n também mantém um conjunto $announcements_n$, no qual são guardadas mensagens de controlo recebidas pelos vizinhos que pertencem ao conjunto $backupPeers_n$. Esta informação é usada para detectar e recuperar de partições nas árvores causadas por falhas ou saídas de nós. De forma a evitar ciclos no envio das mensagens, cada nó mantém ainda

[2] Reactivo significa que o conteúdo da vista parcial mantida pelos nós apenas é actualizada após alterações na filiação do sistema.

um conjunto $receivedMsgs_n$, com os identificadores das mensagens anteriormente recebidas.

Finalmente, de forma a balancear a carga dos nós, i.e., assegurar que a maioria dos nós são apenas interiores numa das árvores e para limitar a carga de reenvio imposta a cada participante, cada nó n mantém uma estimativa da carga de reenvio dos seu vizinhos. Sempre que um nó s envia uma mensagem para outro nó, o primeiro inclui uma lista de valores representando o número de nós para os quais s tem que reenviar mensagens em cada uma das árvores. Dado que esta informação pode ser codificada eficientemente, é incluída em todas as mensagens trocadas entre os nós. Cada nó n mantém a informação mais recente recebida pelo seu vizinho p para cada árvore t numa variável $loadEstimate(p,t)_n$.

Construção das Árvores A criação de cada árvore t é iniciada pelo nó fonte. Para isso, e para cada árvore t, o nó fonte n escolhe aleatoriamente f nós do seu conjunto $backupPeers_n$ e move-os para o conjunto $t.activePeers_n$; estes serão os nós usados pela fonte para encaminhar as mensagens pela árvore t.

Todas as mensagens são marcadas com um identificador único, *muid*, composto pelo par ($sqnb$, t), em que $sqnb$ é um número de sequência e t o identificador da árvore. Os *muid*s de mensagens recebidas anteriormente são guardados em $receivedMsgs_n$[3]. Periodicamente, cada nó n envia uma mensagem SUMMARY com este conjunto para todos os nós em $backupPeers_n$.

Quando um nó n recebe uma mensagem de dados de s por t, primeiro verifica se a árvore já foi criada localmente. A primeira mensagem recebida por uma árvore t inicia o processo de construção de t. O passo de construção para um nó interior é diferente do executado pela fonte. Primeiro, n transfere s de $backupPeers_n$ para $t.activePeers_n$. De seguida, se $\nexists t' : |t'.activePeers_n| > 1$ (i.e., o nó não é interior em nenhuma árvore), então n transfere até $f-1$ nós de $backupPeers_n$ para $t.activePeers_n$. Por outro lado, se n já é um nó interior noutra árvore, o processo pára e n permanece uma folha em t.

De seguida a mensagem é processada. Se a mensagem não é um duplicado, é reencaminhada para os nós em $t.activePeers_n \backslash \{s\}$; caso contrário, o nó transfere s de $t.activePeers_n$ para $backupPeers_n$ e envia uma mensagem PRUNE para s. Após a recepção de uma mensagem de PRUNE, s move n de $t.activePeers_s$ para $backupPeers_s$. Este processo provoca a eliminação do ramo redundante de t.

Executando este algoritmo, os nós tornam-se interiores no máximo numa das árvores. O algoritmo também promove a distribuição da carga (desde que o número de mensagens enviadas através de cada árvore seja semelhante). Por outro lado, dado que os nós escolhidos no processo de construção das árvores são seleccionados de forma aleatória, existe uma probabilidade não desprezável de alguns dos nós não ficarem ligados a todas as árvores. Esta situação é resolvida pelo processo de reparação descrito de seguida.

Reparação das Árvores Os objectivos do mecanismo de reparação são: *i*) assegurar que todos os nós se ligam eventualmente a todas as árvores de disseminação e *ii*) detectar e recuperar de partições na árvore resultantes da ocorrência de falhas. Este componente depende da troca das mensagens SUMMARY entre os nós, tal como descrito anteriormente.

[3] Técnicas para eliminação de informação obsoleta neste conjunto são descritas em[18].

Quando um nó n recebe uma mensagem SUMMARY de outro nó s, este verifica se todos os identificadores recebidos estão presentes em $receivedMsgs_n$. Se nenhuma das mensagens se encontra em falta, a mensagem é descartada. Caso contrário, um par ($muid$, p) é guardado em $announcements_n$. De seguida, para cada árvore t onde uma mensagem se encontra em falta, é activado um temporizador: se as mensagens não forem recebidas quando o temporizador expirar, o nó assume que t ficou quebrada e repara a árvore. O nó n escolhe um vizinho r referente a um par $(muid, p) \in announcements_n$ para reparar a árvore t com base na estimativa $loadEstimate(p, t)_n$ da carga de cada um dos vizinhos. Nomeadamente, r é seleccionado aleatoriamente entre os nós de $announcements_n$ cuja carga esta abaixo de um limite ($maxLoad$) e que são interiores em menos árvores, ou já são interiores na árvore t.

O tempo de espera do temporizador utilizado não deve ser demasiado pequeno de forma a não accionar a recuperação de mensagens devido a pequenos aumentos da latência da rede. No entanto, este valor também não deve ser demasiado grande de forma a garantir que as mensagens são entregues em tempo útil. Tipicamente, este tempo deve ser um factor do RTT da rede.

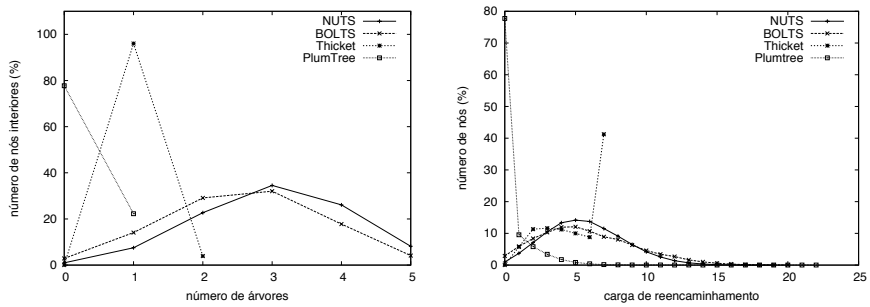
Depois de seleccionar r, o nó n: move r de $backupPeers_n$ para $t.activePeers_n$ e envia uma mensagem GRAFT para r. Esta mensagem inclui a vista que n tem da carga de r (esta informação pode estar desactualizada). Quando r recebe uma mensagem GRAFT de n por uma árvore t, verifica primeiro se n fez a sua decisão baseado em estimativas de carga actualizadas ou se, independentemente da precisão da estimativa, r pode satisfazer o pedido de n sem aumentar o número de árvores onde é interior nem exceder o limite de carga $maxLoad$. Neste caso, r adiciona n a $t.activePeers_r$. Caso contrário, r rejeita o pedido e envia uma mensagem PRUNE a n. Finalmente, se n receber uma mensagem de PRUNE de r, n volta a transferir r de $t.activePeers_n$ para $backupPeers_n$ e tentará reparar t usando um vizinho diferente presente em $announcements_n$.

Reconfiguração das Árvores Os processos descritos anteriormente visam a criação de árvores com cobertura total sobre todos os participantes, onde grande parte dos nós são interiores apenas numa das árvores. Apesar de esta configuração se manter num ambiente estável (sem alterações na filiação do sistema), múltiplas execuções do mecanismo de reparação podem originar situações onde vários nós são interiores em mais que uma árvore.

Para resolver este problema, desenvolvemos um processo de reconfiguração que opera da seguinte forma: quando um nó n recebe uma mensagem de dados não redundante m de um nó s por uma árvore t para a qual n havia já recebido um anúncio por parte de outro vizinho a, n compara a carga estimada de s e a.

Se $\sum_t loadEstimate(s, t)_n > \sum_t loadEstimate(a, t)_n$ e n pode substituir a posição de s na árvore t sem se tornar interior em mais árvores, n tenta substituir a ligação entre s e n por uma ligação entre a e n. Para isso, n envia uma mensagem de PRUNE para s e uma mensagem de GRAFT para a.

Esta alteração apenas acontece se a recepção do anúncio de a ocorrer antes da recepção da mensagem de dados de s. Este facto, garante que este processo contribui para a redução da latência na árvore e evita ciclos. Note que, no caso de um nó atingir o limite de carga $maxLoad$, este será incapaz de ajudar os seus vizinhos no processo de reparação. Por esse motivo, nesse caso o envio de mensagens SUMMARY é cancelado.



(a) Distribuição de nós K-interiores. (b) Distribuição da carga de reenvio

Figura 2. Resultados experimentais para um cenário estável

Alterações na Rede Como referido anteriormente, o protocolo de filiação é responsável por detectar alterações na vista parcial dos nós e notificar o Thicket dessas ocorrências, utilizando as chamadas *NeighborDown*(p) e *NeighborUp*(p). Quando um nó n recebe uma notificação *NeighborDown*(p), este remove p de todas os conjuntos $t.activePeers_n$ e também de $backupPeers_n$. Adicionalmente, todos os anúncios enviados por p são removidos de $announcements_n$. Este processo pode resultar na quebra de algumas árvores. Contudo, o mecanismo de recuperação é capaz de detectar e recuperar dessa situação.

Por outro lado, quando um nó n recebe uma notificação *NeighborUp*(p), p é adicionado ao conjunto $backupPeers_n$. Desta forma, p começará a trocar mensagens SUMMARY com n. Como referido anteriormente, estas mensagens irão permitir que os nós se liguem a todas as árvores e, ainda, balancear a carga dos nós existentes pelos novos nós (utilizando o mecanismo de reconfiguração).

5 Avaliação

Nesta secção são apresentados os resultados experimentais obtidos através de simulações efectuadas no simulador PeerSim[19]. Para isto, desenvolvemos uma implementação do Thicket para este simulador. De forma a obter resultados comparativos, testámos também o desempenho do protocolo Plumtree[6] (que consiste no ponto de partida da nossa solução), assim como as alternativas simplistas discutidas na Secção 3. Todas as abordagens foram executadas sobre a mesma rede sobreposta, mantida pelo protocolo HyParView[16]. Este protocolo é capaz de recuperar de cenários em que 80% dos nós falham simultaneamente. Como o HyParView utiliza o protocolo TCP para manter os vizinhos da rede (nomeadamente para detectar falhas), o nosso sistema não contempla perdas de mensagens.

Testámos todos os protocolos primeiramente num ambiente estável, onde não foram induzidas falhas, e, posteriormente, em cenários com falhas. No segundo caso, avaliámos a fiabilidade do processo de difusão na presença de falhas sequenciais de nós. O leitor poderá também encontrar resultados que avaliam a capacidade de reconfiguração do Thicket em cenários catastróficos, em que 40% dos nós falham simultaneamente[17]. De seguida, descrevemos a configuração experimental utilizada durante as experiências.

5.1 Configuração Experimental

O progresso das simulações é expresso em ciclos (utilizando o motor baseado em ciclos do simulador). Cada ciclo corresponde a 20s. Em cada ciclo, a fonte difunde

T mensagens simultaneamente, uma por cada árvore existente (no caso do Plumtree, que constrói apenas uma árvore, todas as T mensagens são enviadas através dessa mesma árvore). Como referido anteriormente, assumidos que as ligações são perfeitas, contudo as mensagens não são entregues instantaneamente, em vez disso consideramos os seguintes atrasos no envio das mensagens (estes atrasos foram implementados utilizando o motor de eventos do simulador[4]):

Atraso no Emissor Assumimos que cada nó possui um limite de largura de banda de saída. Isto permite simular congestão no envio de dados quando um nó necessita de enviar várias mensagens consecutivamente. Em particular assumimos que cada nó pode transmitir $200K$ bytes/s. Assumimos também que o conteúdo das mensagens de dados ocupa 1250 bytes, enquanto que as mensagens SUMMARY, ocupam 100 bytes.

Atraso na Rede Assumimos que são introduzidos atrasos adicionais na rede. Mais concretamente, durante as simulações uma mensagem transmitida sofre um atraso de valor aleatório entre 100 e 300 ms. Estes valores foram seleccionados tendo em conta medições de latência realizadas na infra-estrutura PlanetLab[5].

As experiências foram realizadas utilizando uma rede de 10.000 nós e todos os resultados correspondem à agregação de 10 execuções independentes de cada experiência. Todos os protocolos testados, com a excepção do Plumtree, foram configurados para gerar $T = 5$ árvores. Adicionalmente, o Thicket estabelece árvores usando um fanout $f = 5$ e a abordagem NUTS inicia o conjunto de nós *eager* com 5 vizinhos escolhidos aleatoriamente. O Thicket, o Plumtree, e o NUTS operam sobre uma rede não estruturada com grau 25, enquanto que cada uma das 5 redes sobrepostas utilizadas pelo BOLTS possui um grau de 5. Além disso, configurámos o limite de carga de reencaminhamento máxima por nó (parâmetro *maxLoad*) com o valor 7. Este valor deve ser suficientemente grande para garantir que cada nó possui o número suficiente de filhos necessários para a construção das múltiplas árvores. Por outro lado, não deve ser demasiado grande de forma a limitar a carga de reencaminhamento dos nós. Admitindo a construção óptima das árvores, o valor ideal para este parâmetro seria 5, dado pelo quociente entre o tamanho da vista parcial de cada nó (25) e o número de árvores a construir ($T = 5$). Porém, como o nosso protocolo é uma aproximação, determinámos experimentalmente que o valor 7 é o menor que permite a construção das várias árvores. O temporizador iniciado após a recepção de um anúncio foi configurado para um valor de 2s que, pretendendo ser um valor reduzido, não desencadeia recuperações desnecessárias de mensagens. No protótipo desenvolvido, é enviada uma mensagem SUMMARY assim que a respectiva mensagem de dados é entregue, contendo apenas o identificador dessa mensagem.

Todas as experiências começam com um período de estabilização de 10 ciclos, que não são considerados nos resultados apresentados. Durante estes ciclos, todos os nós se juntam à rede sobreposta e a topologia da rede estabiliza. Após este período, inicia-se o processo de difusão; este desencadeia o processo de construção das árvores.

[4] A unidade de tempo mínima do sistema é 1ms.

[5] As medições podem ser encontradas em `http://pdos.csail.mit.edu/~strib/pl_app/`

5.2 Ambiente Estável

Primeiro, analisámos as medidas de desempenho relevantes para o Thicket num ambiente estável onde não ocorrem falhas de nós. Começamos por avaliar a distribuição de nós de acordo com o número de árvores em que estes são interiores. Os resultados são exibidos na Figura 2(a). O Plumtree mostra o ponto de partida num cenário onde existe apenas uma única árvore. Note-se que, com uma única árvore, apenas 21% dos nós são interiores, e 79% são folhas.

Usando ambas as estratégias NUTS e BOLTS, apenas uma pequena fracção (abaixo dos 20%) dos nós são interiores numa única árvore (repetimos aqui a imagem da Secção 3 para conveniência do leitor). Para ambas as abordagens, existe ainda uma pequena percentagem de nós que são interiores em todas as 5 árvores. Como referido anteriormente, este facto motiva a necessidade de algum tipo de coordenação durante o processo de construção das árvores.

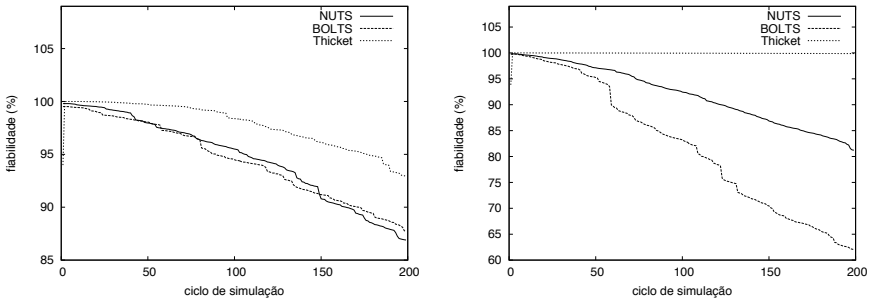
Porém, no Thicket praticamente todos os nós são interiores em apenas uma das árvores. Uma fracção mínima (cerca de 1%) permanece interior em 2 árvores. Este é um efeito secundário do mecanismo de recuperação, que garante a cobertura de todas as árvores de disseminação. Ainda assim, nenhum nó (com a excepção da fonte) é interior em mais que 2 árvores. Este facto valida o desenho do Thicket. Adicionalmente, quase nenhum nó é folha em todas as árvores; contribuindo para a fiabilidade do processo de difusão (ver resultados abaixo), assegurando uma distribuição de carga uniforme entre os participantes, permitindo também uma utilização mais eficiente de todos os recursos presentes no sistema.

A Figura 2(b) mostra a distribuição da carga de reencaminhamento do sistema *i.e.*, a distribuição dos nós de acordo com o número de mensagens que estes devem reenviar através de todas as árvores. Devido ao facto do Thicket limitar a carga de cada nó durante os processos de construção e manutenção das árvores, nenhum participante excede o limite de 7 envios no total de todas as árvores em que este é interior (normalmente 1 como explicado anteriormente). Adicionalmente, mais de 40% dos nós reencaminham a quantidade máxima de mensagens, com mais de 55% dos nós a reencaminhar um reduzido número de mensagens. As restantes soluções, possuem valores de carga muito variáveis, com vários nós responsáveis pela transmissão de mais de 10 mensagens e alguns com cargas superiores a 15 mensagens. Note que o Thicket é o único protocolo onde quase nenhum participante tem uma carga de reenvio de 0. Este facto demonstra os benefícios associados à utilização de recursos e distribuição de carga conseguidos pelo Thicket.

5.3 Tolerância a Faltas

Nesta secção estudámos o impacto de falhas sequenciais na fiabilidade do processo de difusão usando o Thicket, o NUTS, e o BOLTS. Nas nossas experiências o nó fonte e os nós raízes das árvores do NUTS nunca falham.

Considerámos a fiabilidade, assumindo que o processo de difusão utiliza as várias árvores para introduzir redundância nos dados disseminados (utilizando, por exemplo, técnicas de *network coding*). Desta forma, assumimos que para cada segmento de 5 mensagens enviadas (uma por cada árvore), se um nó receber pelo menos 4 das mensagens, é capaz de reconstruir totalmente o segmento de dados, caso contrário consideramos que o nó falha a recepção do segmento. Definimos



(a) Falha de Nós Aleatória. (b) Falha de Nós Dirigida.

Figura 3. Resultados experimentais para um cenário catastrófico.

fiabilidade como sendo a percentagem de nós capazes de reconstruir os segmentos de dados enviados.

Depois de um período de estabilização (5 ciclos) configurámos o nó fonte para enviar um segmento de dados por ciclo. Em cada ciclo, induzimos também uma falha num dos nós. Medimos a fiabilidade do processo de difusão no final de cada ciclo da simulação. A selecção do nó que falha em cada ciclo foi efectuada usando duas políticas distintas: *i*) seleccionado o nó aleatoriamente; *ii*) seleccionado o nó aleatoriamente de entre os nós que são interiores em mais árvores. Não permitimos que os nós tomassem medidas de recuperação durante as simulações.

A Figura 3 exibe os resultados para ambos os cenários. Quando seleccionamos os nós a falhar aleatoriamente (Figura 3(a)) a fiabilidade do Thicket decai lentamente. Isto acontece porque a maioria dos nós é apenas interior numa das árvores. Por isso, cada falha afecta apenas nós abaixo da falha numa única árvore. Devido ao facto dos nós serem capazes de reconstruir o segmento de dados mesmo sem receberem uma das mensagens enviada por uma das árvores, a maioria consegue ainda assim reconstruir os segmentos de dados desde que se mantenham ligados a (pelo menos) 4 árvores. A fiabilidade decresce mais acentuadamente nas abordagens NUTS e BOLTS. Isto acontece devido à grande quantidade de nós que são interiores em mais do que uma árvore, o que contribui para que a falha de um único nó afecte o fluxo de dados de várias árvores.

O Thicket é também extremamente robusto face a falhas direccionadas aos nós interiores num maior número de árvores (Figura 3(b)), e a sua fiabilidade permanece constante em 100%. Isto acontece devido ao seguinte fenómeno: ao ser imposto um limite de carga a cada nó do Thicket, os nós que são interiores em mais que uma árvore são responsáveis por enviar um pequeno número de mensagens por cada árvore. Desta forma, o número efectivo de nós afectados na árvore onde o nó que falha é interior é menor. Além disso, porque as ligações nunca são usadas em mais de uma árvore, este grupo de nós é disjunto, e consequentemente podem ainda receber mensagens das restantes 4 árvores. Por outro lado, o NUTS e o BOLTS são severamente afectados por este cenário devido ao facto que alguns nós serem interiores em todas as árvores.

6 Conclusões

Neste artigo apresentámos o Thicket, um algoritmo totalmente descentralizado para a construção e manutenção de múltiplas árvores, nas quais cada nó é interior em apenas uma ou num número reduzido de árvores, numa rede não estruturada.

O Thicket permite a distribuição da carga do sistema por todos os nós e, ainda, o envio de dados redundantes por diferentes árvores de forma a tolerar falhas de nós ou ligações.

Referências

1. Liang, J., Ko, S.Y., Gupta, I., Nahrstedt, K.: MON: on-demand overlays for distributed system management. In: Proceedings of WORLDS'05. (2005)
2. Huang, Y., Fu, T.Z., Chiu, D.M., Lui, J.C., Huang, C.: Challenges, design and analysis of a large-scale p2p-vod system. ACM SIGCOMM Comp. Comm. Review **38**(4) (2008) 375–388
3. Eugster, P.T., Guerraoui, R., Handurukande, S.B., Kouznetsov, P., Kermarrec, A.M.: Lightweight probabilistic broadcast. ACM TOCS **21**(4) (2003) 341–374
4. Birman, K.P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y.: Bimodal multicast. ACM Transactions on Computer Systems (TOCS) **17**(2) (1999)
5. Frey, D., Murphy, A.L.: Failure-Tolerant Overlay Trees for Large-Scale Dynamic Networks. In: Proceedings of P2P'08, Washington, DC, USA, IEEE Computer Society (2008) 351–361
6. Leitão, J., Pereira, J., Rodrigues, L.: Epidemic Broadcast Trees. In: Proceedings of SRDS'07, Beijing, China (2007) 301–310
7. Rowstron, A.I.T., Kermarrec, A.M., Castro, M., Druschel, P.: SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In: Net. Group Communication. (2001) 30–43
8. Allani, M., Leitão, J., Garbinato, B., Rodrigues, L.: RASM: A Reliable Algorithm for Scalable Multicast. In: Proc. of Euromicro PDP'2010, Italy, INESC-ID (2010)
9. Chou, P.A., Wu, Y.: Network Coding for the Internet and Wireless Networks. IEEE Signal Processing Magazine **24**(5) (Setembro 2007) 77–85
10. Johnson, D.S., Lenstra, J.K., Rinnooy, H.G.: The complexity of the network design problem. Networks **8**(4) (1978) 279–285
11. Castro, M., Druschel, P., Kermarrec, A.M., Nandi, A., Rowstron, A.I.T., Singh, A.: SplitStream: high-bandwidth multicast in cooperative environments. In: Proceedings of SOSP'03, New York, NY, USA, ACM (2003) 298–313
12. Venkataraman, V., Yoshida, K., Francis, P.: Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast. In: Proceedings of ICNP '06, Washington, DC, USA, IEEE Computer Society (2006) 2–11
13. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Proceedings of Middleware '01, London, UK, Springer-Verlag (2001) 329–350
14. Zhuang, S.Q., Zhao, B.Y., Joseph, A.D., Katz, R.H., Kubiatowicz, J.D.: Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In: Proc. of NOSSDAV'01. (2001)
15. Padmanabhan, V.N., Wang, H.J., Chou, P.A., Sripanidkulchai, K.: Distributing streaming media content using cooperative networking. In: Proceedings of NOSSDAV '02, Miami, Florida, USA, ACM (2002) 177–186
16. Leitão, J., Pereira, J., Rodrigues, L.: HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast. In: Proc. of DSN'07, UK (2007) 419–429
17. Ferreira, M., Leitão, J., Rodrigues, L.: Thicket: A protocol for building and maintaining multiple trees in a p2p overlay. Technical Report 28, INESC-ID (May 2010)
18. Kaldehofe, B.: Buffer management in probabilistic peer-to-peer communication protocols. In: Proc. of SRDS'03, Florence,Italy, SRDS (2003) 76–85
19. Jelasity, M., Montresor, A., Jesi, G.P., Voulgaris, S.: The Peersim Simulator

Towards full on-line deduplication of the Web

Ricardo Filipe and João Barreto

Inesc-ID/Technical University Lisbon
[ricardo.filipe,joao.barreto]@ist.utl.pt

Abstract. The Internet is widely used nowadays but still has important limitations. While average Web page size keeps increasing, the bandwidth available to each user is not growing at the same rate. Users want to do several things at the same time on the Web and they don't want to wait much time to do it.
We notice that most Web pages have substantial redundancy with previous versions of themselves and other pages. With this in mind, we developed a novel deduplication system for HTTP traffic.
Our proposed system works from end client to server and only for text resources transmitted through the HTTP protocol. Our system transfers less bytes than a plain HTTP transfer, achieving gains of 82% when downloading pages from popular sites such as `cnn.com`. It completes a transfer request 4% faster than plain HTTP. It is also not as time consuming as delta-encoding between the requested resource and an older version of it. Our approach is the first to be implemented and analysed on a real Internet environment.

1 Introduction

The Web has had a major growth in recent years. The number of users keeps growing, the average Web page size and number of objects per page have steadily increased [1], the data available through the Internet is doubling every year [2].

This enormous growth entails many challenges. While available bandwidth has steadily grown in some network environments, it remains a scarce resource in others, such as Bluetooth networks. Even in high bandwidth scenarios, each transferred byte frequently has a cost, for instance in terms of battery consumption or fees paid to the client's ISP, which clients naturally wish to minimize.

Fortunately, the substantial redundancy that exists across the contents downloaded from the Web tells us that most of the transmitted payload could be, in fact, avoided. Most individual files exhibit intra-file redundancy, which one can eliminate with compression schemes like Gzip. Furthermore, Web browsers frequently request resources (e.g. previously downloaded HTML pages or images) that were already downloaded by the same client (or nearby clients) and have not changed meanwhile, a well studied problem that Web caches effectively tackle.

However, there is evidence that much redundancy in the Web originates from situations that neither per-file data compression nor Web caching can exploit. Such redundancy is caused by two phenomenons. A first one occurs when the

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 291–302

same resource is referenced by different URLs, which occurs mostly in image resources and is commonly called aliasing [4]. Most importantly, the most common phenomenon is *resource modification* [3], when a resource changes by little pieces over time. We see this everyday in our social Web. News sites are updated several times per day, they let you comment on those news and see other people's comments. Social networks let you comment on other people's profiles. Blogs make it possible to post new content and have it commented too. Forums and discussion sites have threads of responses from their users. Hence, pages loaded at different times will often be very similar in content, with marginal changes introduced by the new pieces of information.

In order to exploit the above phenomena, we need techniques for full *deduplication* [5] of the Web. In other words, techniques that are free from the limitations of data compression and classical Web caching, thus able to eliminate any form of redundancy: be it within the individual resource, or across different resources (or across distinct versions of the same resource); going from the granularity of the individual resource down to much finer-grained chunks of redundant information.

Perhaps that older deduplication technique that breaks the above mentioned limitations is delta encoding [3]. In delta encoding, a client that already holds some version of a given resource and requests a newer version will only receive a compact set of differences from the latter to the former, which the client should then apply upon its local (older) version. Delta-encoding is still very much in use, mostly when the deltas can be precomputed, which is often the case when distributing software patches and service packs. However, since the algorithms used for computing deltas are typically very time-expensive [3], delta encoding is not suitable for distributing dynamic content like most Web pages that are created on-the-fly.

More recently, much research has proposed techniques for on-line deduplication of dynamic Web content. However, the proposed techniques exhibit important drawbacks that severely limit their usefulness in the large-scale Web. Some assume strong synchronization between client and server state [6], while the memory requirements of others do not scale well to large-scale Web servers (e.g. [9], [8]). Some are only able to detect redundancy across the contents transmitted from the ISP proxy to its clients, neglecting the proxy-server path [11]. More importantly, most solutions have never been implemented nor evaluated experimentally (e.g. [6], [8]).

In this paper we present a novel deduplication system we called dedupHTTP. The proposed system works from the end client to the origin server. It detects redundancy in text resources transferred through the HTTP protocol. It uses the client's cached files from the requested resource's domain as reference resources. It eliminates redundant transfers originating from resource modification and aliasing of resources inside each domain. The current implementation is browser and server transparent and has a very lightweight communication protocol. Our results show that dedupHTTP:

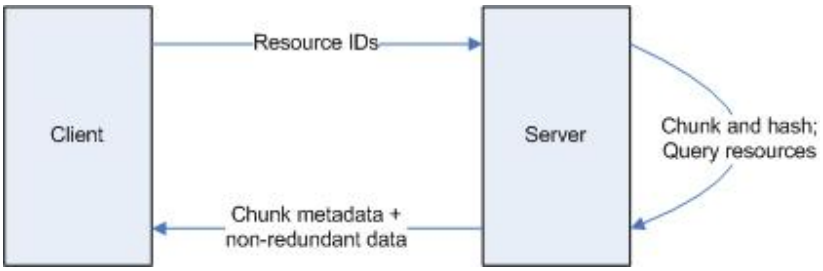


Fig. 1. Architecture Overview

- Outperforms plain HTTP transfer in the number of bytes transferred by 82% and time consumed by 4%, at large bandwidth;
- Has 10% better time performance than delta-encoding with a previous version of the requested resource;

The rest of the paper is structured as follows. First we will explain the resource division algorithm into chunks. Then we will overview the architechture of the system and the communication protocol. We then show the results of our tests, comparing our system with regular transfers, gzip compression and a simple delta-encoding system for Web transfer. Finally we go through some of the related work and draw our conclusions to this work.

2 Architecture overview

The dedupHTTP system has two implementation points, we'll call them client and server for simplicity. The client stores the resources and maps them by host domain. When it makes a request for a resource the client fetches all the identifiers from stored resources of the same host domain as the requested resource. These identifiers represent the reference resources for the request.

When the server responds to a request it divides the response into chunks using the algorithm from Section 4. Those chunks' hashes are stored on the server along with their offset within the resource and their length (16 bytes metadata per chunk on the current implementation). The server then retrieves the client resource identifiers from the HTTP header. The server searches for the response resource's chunks in the identified client resources, and the metadata required for resource reconstruction is sent to the client.

When the client receives the response it starts reconstructing the requested resource by fetching the referenced chunks from local cache. Then it combines them with the non-redundant content on the response in the correct order. The resource is saved on the client and mapped to its host domain.

3 Protocol

When a new request is done by the client, the reference resources identifiers are serialized into a byte array and placed inside a new HTTP header called "Versions", which goes on the request header. This accomplishes two important goals. First there is no need for the server to keep client based state, as it is each client's responsibility to provide the right list of reference resources. Secondly, it allows the client cache to have the regular resource cache eviction policies, the resources the client tells the server are in its cache are the ones the server will use as reference resources. [1]

After dividing the resource, the corresponding chunk meta information is stored on the server, mapping the resource. The server goes through all chunks' hashes of the requested resource. For each such chunk the server queries each of the client's reference resources for the chunk's existence. If the chunk is not found the server also queries the current response's respective resource. This way we not only detect redundant chunks between different resources but also inside the resource being served. Whenever a chunk is found on a queried resource we only need to tell the client where to find it. If the chunk is not found in any of the queried resources, we copy the chunk data to the final response.

The final response begins with a metadata section. The size of this section is stored on a new HTTP header we called "Metadata". In this section goes all the information needed for the client to find the redundant chunks in its cache. Each redundant chunk is here identified by a four-part tuple: the offset in the current response where the chunk is to be appended, the resource identifier where the client can find the chunk content, the offset of the resource where the chunk content can be found and the length of the chunk. At the end of the metadata block we append the non-redundant response content.

When the client receives the response it only has to go through the metadata, copy to the final response the chunks referenced in the metadata, from the locally cached resources, and copy the non-redundant content from the received response to the final response, in the respective order. Thus, the new resource is reconstructed on the client and the client does not even have to store meta information about the chunks, only the resource identifier and the resource itself.

Each response has an identifier created on the server, which is sent in the metadata block. We have opted to store this value in 2 bytes only, which helps shorten the metadata block. This means there can only be 65536 different resources referenced in the server. We believe this is more than enough for a regular server, even with dynamic resources generated on-the-fly, since after some days or weeks the older resources metadata can be evicted. Even if the client's cache could accommodate for resources that were older than that, the usefulness of such resources is expectably low. Redundancy with fresh downloaded contents

[1] As we show next, each client will be handling look-ups to its list of reference resources. At the same time, it can try to evict some of such entries. We can easily ensure safe synchronization of such accesses by using regular read-write locks.

should mostly come from recent versions, rather than older ones, which we intend to explore in future work.

For this we use an age value on each resource. When the age value for the resource has been reached, if the client still has the resource cached it is evicted from the client's cache. At the same time the server will evict the chunk metadata it possesses for that resource. This way the client and server caches are kept synchronized without additional network messages.

The other pieces of the metadata are all stored in 4 byte integers which makes for a 14 bytes total of metadata to reference a chunk. This number should be taken into account when choosing the chunk size, since there will be no gain if there can be chunks smaller than this.

3.1 Optimizations

We have seen there are consecutive chunks detected in the same resource many times. Since we check the client's resources in the same order for every chunk, we can save on metadata by regarding the sequence of consecutive chunks as a single chunk. Hence, we only send a meta-data block for the large coalesced chunk (instead of one block per consecutive redundant chunk). A particular case where such optimization is very effective is when a requested resource is aliased on the domain; the only thing that will be sent to the client is a chunk's worth of metadata with the whole resource length.

The same resource can be requested by different clients. On the first request the algorithm will be the one explained in this section. For the second and subsequent client requests we can save an important step of processing, the chunk division algorithm.

The server stores all served request's ETags. These are unique resource identifiers present in the HTTP specification. When a new request is to be served we check if the ETag has already passed before. If it has, we retrieve the corresponding resource chunk metadata and continue to the next phase. Only the first request to each resource has to go through the chunk division phase.

4 Chunk division algorithm

The first step to take on the server when a resource is requested from a client is to divide that resource into indexed chunks. This way we can easily identify which parts of the resource the client already has and which are new and have to be sent.

We start by creating per-byte hashes of the resource content. These hashes represent smaller chunks of the resource. This is done by using the rolling hash function of Karp-Rabin fingerprinting [13]. Let c_i be the byte at index i of the resource stream and k be the length of the Karp-Rabin chunk. Let b be a prime number as base. The hash of the Karp-Rabin chunk is given by:

$$H(c_i...c_{i+k}) = c_i * b^{k-1} + c_{i+1} * b^{k-2} + ... + c_{k-1} * b + c_k$$

Since b is a constant, the iteratibility of this function allows us to calculate the next byte's hash with some simple operations on the previous byte's hash with the new byte:

$$H(c_{i+1}...c_{i+1+k}) = ((H(c_i...c_{i+k}) - c_i * b^k) + c_{k+1}) * b$$

Now we have to select the hashes that represent the resource. We could select all hashes, but that is unfeasible memory-wise since we have an hash per byte of content. So, we select certain hashes to be the boundaries of larger chunks of data.

For this step we chose the Winnowing technique since it has been experimentally proven to give the better redundancy detection [7]. We enforce a minimum and maximum chunk size because these content based methods can create too big or too small chunks compared to the desired size.

After selecting the chunk boundaries we hash each larger chunk using a 64 bit MurmurHash [15]. This choice was driven by the fact that MurmurHash outperforms cryptographic hash functions such as MD5 or SHA1, while retaining a very low collision rate. We do not need cryptographic security in our hashes, so using MD5 or SHA1 would be overkill. They are much slower and offer similar collision rates (in fact MurmurHash offers better hash distribution than MD5 and very similar to SHA1 [15]).

5 Implementation

Our system was implemented with a proxy for the client machine and a reverse proxy for the server machine. Both proxies were implemented using the OWASP Proxy [16] library, which takes care of every aspect in HTTP communication. Each proxy was extended with the proper functionalities described in the algorithms and architecture to deal with the resource content. This was done in about five hundred lines of Java code, since that was the language of OWASP Proxy. The MurmurHash code was used from a public library [17] that ported the hash code to Java.

Using proxies for the implementation allows us to be browser and server independent, not having to familiarize with a unique browser and server implementation, nor favor one over another. It has the inconvenients of adding proxy latency to the connection and having it's own cache on the client side. The proxy cache and browser cache will certainly overlap in most of their data. On the server side this is not an issue since we only store resource metadata.

The resource metadata is stored per chunk on the server. It is composed of two 32 bit integers, offset in the resource content and chunk length, and a 64 bit long for the chunk hash.

6 Evaluation

The test server machine is an Intel Pentium 4 @ 3.20 GHz with 2 GB of RAM, while the client machine is an Intel Core 2 Duo @ 2.16 GHz with 4 GB of RAM.

Number of files	Total Size	Server memory overhead
337	41,5 MB	16 bytes * Number of Chunks

Table 1. Workload specification

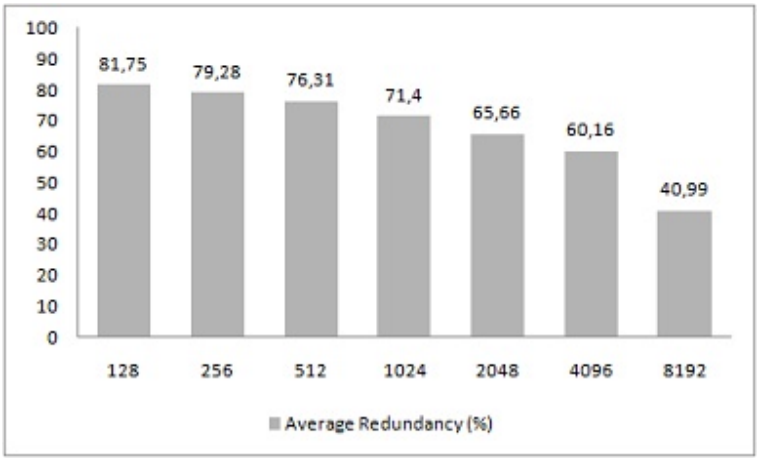


Fig. 2. Experiment average redundancy for several chunk sizes

The workload for our experiments was created by downloading every news and comments page from www.cnn.com (Table 1). These pages change very frequently which makes them a perfect fit for a system like this. We did this in two consecutive days so that we could compare the redundancy that remains from one day to the other, mimicing a regular user that likes to read the news every day. We removed unchanged pages from the second day, since they would be treated by a regular browser cache.

Our chunk division algorithm has two parameters that should be experimented for the best redundancy detection: the smaller Karp-Rabin fingerprint length (Table 2) and the larger chunk length (Figure 2).

The large chunks were experimented with sizes ranging from 128 bytes up to 8192 bytes (8 KB). The smaller the chunk size the more metadata the server has to store, since we have more chunks per file. Since our metadata is not too heavy we decided the 128 bytes chunks should be used for the rest of the experiments. The increase in number of chunks to search for could factor in the choice of chunk size. Although, this has not been noticeable in our experiments, so we disregard it.

We have experimented the Karp-Rabin fingerprint sizes of 16, 32 and 48 bytes. The 16 bytes fingerprints give a better redundancy detection, although by a small margin, so we decide to use it for the remainder of the experiments. This parameter has no influence in the number of chunks nor in the complexity of the division algorithm, so the value with better results should be chosen.

Size of Karp-Rabin fingerprint	Redundancy detected
16	81,75%
32	80,64%
48	80,74%

Table 2. Redundancy detected with varying Karp-Rabin fingerprint size

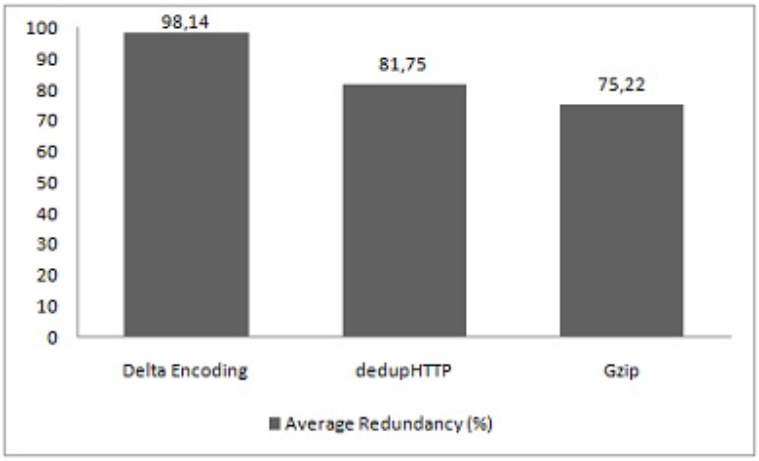


Fig. 3. Experiment average redundancy for several solutions

We compared our solution with three others. The first one is plain HTTP transfer, then compressing the resources with Gzip and finally we tested against a delta encoding solution we created.

This delta encoding system stores the resources on the server and client. If the client is downloading a resource that already exists in its cache, but is outdated, the server creates a delta between both versions of the resource on-the-fly. Then it sends the delta to the client and it reconstructs the requested resource. This solution does not work for the first request of a resource, so we have not included those results in our measures.

We want to see how does dedupHTTP compare to the other solutions in redundancy detected and Time to Display (Figures 3 and 4). Time to Display is the time it takes since the resource is requested in the client until it is displayed on the client's machine. If the deduplication algorithm takes more processing time than that which is spared by less transfer traffic, the Time to Display will worsen which is undesirable. Our Time To Display results are the mean average time of downloading all of the workload's files one by one.

Delta encoding detects most existing redundancy that we could hope to detect, since it is a lossless algorithm that works with only previous versions of the same resource. This should be the comparison point of our solution as far as redundancy detection goes. On the other hand it is a very costly algorithm, which brings very high Time to Display. Plain HTTP transfer should be the ref-

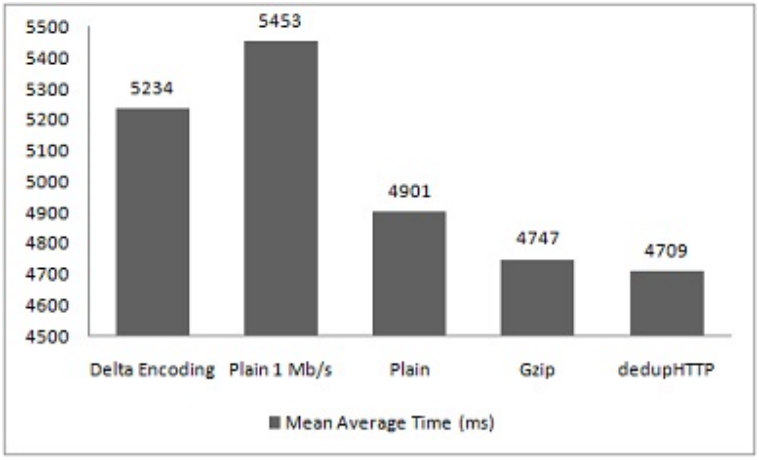


Fig. 4. Experiment average Time to Display for several solutions

erence point here, since we want to improve on the every day solution's Time to Display. We have also compared to plain HTTP transfer on limited bandwidth (1 Mb/s) to assert that where fewer bandwidth is available our system would be most benefitial.

As far as redundancy detection is concerned, we outperform Gzip by a meaningful margin. Comparing to delta encoding we are a bit far from detecting as much redundancy, although we didn't isolate only the responses that could be delta-encoded to get dedupHTTP results. If we did that our redundancy detection is much closer to delta encoding (95%) and plus we still detect much redundancy on the first time resource requests, where delta-encoding cannot be used.

Regarding Time to Display we outperform every single solution we have tested against, achieving comparable results to Gzip encoding. This is very significant as we prove our system's feasibility to complement regular Web caching. We can also confirm that using delta encoding on-the-fly is much more computationaly heavy than our system.

7 Related work

Chunk fingerprinting has been widely used in distributed file systems [10] and other deduplication systems for the Web (e.g., [11]). For chunk boundary selection these systems used the scheme proposed by Manber [12], but as Anand et al. [7] have shown the Winnowing [14] technique is more profitable in this regard.

Manber's fingerprint selection scheme selects each hash that complies with $H \bmod P = 0$ where P is the desired larger chunk size. On average they expect to have an hash selected in P bytes as a chunk boundary. Winnowing differs by selecting the smaller hash in each consecutive window of P bytes.

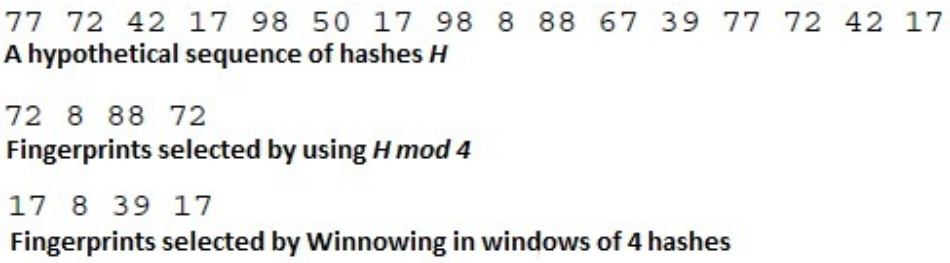


Fig. 5. Manber vs. Winnowing: chunk boundary selection

There have been many systems proposed for Web deduplication, each with its own limitations. Spring and Wetherall [6] proposed a shared cache architecture where two caches store chunks of data in the same way. When redundant content is identified by the transmitting cache only the corresponding chunks fingerprints are sent. They didn't implement this architecture, it was used as an example of where to use their redundancy detection algorithm. They also do not address how to keep both caches synchronized. They suggest this approach for a protocol independent system, but in their results we can see that only HTTP is useful to analyse since it is the most used protocol and with most redundant data too.

Anand et al. [7] have recently confirmed these results for an enterprise setting (on a university setting P2P traffic is a bit more relevant). They also suggest that an end-to-end approach is preferable to a middlebox one since most users do not share the same surfing habbits, therefore the gain of detecting cross-user redundancy would be little when compared to the cost of detecting redundancy across more data. We applied these studies results to devise dedupHTTP.

Chan and Woo [8] presented a general approach to cache based compaction. It includes two main ideas: a selection algorithm to choose reference objects, and an encoding/decoding algorithm that acts upon a new object using the selected reference objects. They proposed a dictionary based solution for encoding/decoding. When there are no reference resources it would act as gzip and when there are some it would get redundant strings from them. Their approach is not scalable, if the number of reference objects increases the complexity of the algorithm increases at the same time. While our approach also increases in complexity with the increase of reference objects, it is much less noticeable. They have also not implemented and tested their proposal. Their selection algorithm though gave interesting results. They tell us that resource redundancy is not limited to previous versions of that resource. Most resources in the same folder and even in the same domain still have much redundancy to be taken advantage of as reference resources for deduplication.

Banga et al. [9] developed an optimistic deltas system. The server stores the resources it sends to the clients. When a new version of a resource is requested it creates a delta between the new version and the version the client has and sends it. When a resource is to be sent for the first time to a client, an older version

is immediately sent when the request is received on the server. Then if the new resource is the same as the sent one the only thing necessary is to respond to the client saying that. If it is different a delta is created between the two versions and sent. The system is optimistic since it expects to have enough idle time to send the old version of the resource while the new version is being created. It also assumes the changes between the two versions are small relatively to the whole document. With the bandwidths of today this kind of latency reduction would be inefficient, more often than not overlapping the transport of the older resource with the response with the new one.

Rhea et al. developed a Value Based Web Cache (VBWC) [11]. It is a system similar to Spring and Wetherall's but they implemented and tested it. They aimed to use it at the ISP proxy, where it would store the chunk's hashes that went to the clients. The proxy did not store the actual data. When a chunk goes to a client, the proxy cache checks if that client already has that chunk. If it does it only transfers its fingerprint. The bandwidth savings of VBWC are only on the ISP proxy-client connection. There is no cross-client redundancy detection since their hash cache is mapped by client. Since each client also stores the chunk's hashes they are redundant in the ISP proxy for all clients that requested the same chunks. All of these problems could be solved if the hash cache was implemented at the origin server. Our approach is an example of how this can be implemented.

8 Future work and Conclusions

This work presents evidence that suggests that current HTTP transfer systems can be much improved with simple changes on both client and server side and fosters further research on this topic, which has been stagnant for quite some time. Even if bandwidth is ever less a concern for desktop users, we must remind ourselves we live in a wireless world where bandwidth is still a scarce resource.

We have devised a new on-line deduplication system for the Web. We have shown that it outperforms the solutions available to current Web servers, including plain HTTP transfer and Gzip compression. We are the first to implement and test such a system in real internet conditions.

There are several items we can work further in our system. We plan to test how does redundancy between two versions vary over time, instead of just two days test up to thirty days difference. This will help us determine at what point in time we should evict resource metadata on the server and client.

We want to test an hybrid algorithm that uses our system and then compresses the responses with Gzip. It should yeld even better redundancy detection, although it may cost more Time to Display than affordable.

We plan to test the system in a controlled network environment, with the client and server nodes interconnected by a network link with tunable bandwidth and latency, in order to avoid external arbitrary interferences on those parameters.

We also plan to run more detailed benchmark tests, which will allow us to understand the factors contributing to the latency of our solution; namely, time

spent on Web server CPU processing, message handling, message transmission, among others. From these results, we should be able to further optimize our solution and evaluate its true scalability.

References

1. http://www.websiteoptimization.com/speed/tweak/average-web-page/
2. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html
3. Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In Proc. of ACM SIGCOMM, 1997
4. Terence Kelly , Jeffrey Mogul, Aliasing on the world wide web: prevalence and performance implications, Proceedings of the 11th international conference on World Wide Web, May 07-11, 2002, Honolulu, Hawaii, USA
5. Nagapramod Mandagere , Pin Zhou , Mark A Smith , Sandeep Uttamchandani, Demystifying data deduplication, Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion, December 01-05, 2008, Leuven, Belgium
6. Neil T. Spring , David Wetherall, A protocol-independent technique for eliminating redundant network traffic, Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, p.87-95, August 28-September 01, 2000, Stockholm, Sweden
7. Ashok Anand , Chitra Muthukrishnan , Aditya Akella , Ramachandran Ramjee, Redundancy in network traffic: findings and implications, Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems, June 15-19, 2009, Seattle, WA, USA
8. Mun Choon Chan and Thomas Y. C. Woo, Cache-based compaction: A new technique for optimizing web transfer, Proceedings of IEEE INFOCOM, March 1999
9. Gaurav Banga , Fred Douglis , Michael Rabinovich, Optimistic deltas for WWW latency reduction, Proceedings of the USENIX Annual Technical Conference, p.22-22, January 06-10, 1997, Anaheim, California
10. Athicha Muthitacharoen , Benjie Chen , David Mazires, A low-bandwidth network file system, Proceedings of the eighteenth ACM symposium on Operating systems principles, October 21-24, 2001, Banff, Alberta, Canada
11. Sean C. Rhea , Kevin Liang , Eric Brewer, Value-based web caching, Proceedings of the 12th international conference on World Wide Web, May 20-24, 2003, Budapest, Hungary
12. Udi Manber. Finding similar files in a large file system. In Proceedings of the USENIX Winter 1994 Technical Conference, pages 110, San Fransisco, CA, USA, 1721 1994
13. Karp, R. M. and Rabin, M. O. 1987. Efficient randomized pattern-matching algorithms. IBM J. Res. Dev. 31, 2 (Mar. 1987), 249-260
14. Saul Schleimer , Daniel S. Wilkerson , Alex Aiken, Winnowing: local algorithms for document fingerprinting, Proceedings of the 2003 ACM SIGMOD international conference on Management of data, June 09-12, 2003, San Diego, California
15. A. Appleby, MurmurHash 2.0, http://sites.google.com/site/murmurhash/, 2009
16. R. Dawes, OWASP Proxy, http://www.owasp.org/index.php/Category:OWASP_Proxy, 2010
17. V. Holub, Java implementation of MurmurHash, http://d3s.mff.cuni.cz/ holub/sw/javamurmurhash

Computação Gráfica

Construção Interactiva de Exposições Virtuais

Jorge Carvalho Gomes, Maria Beatriz Carmo, Ana Paula Cláudio

Faculdade de Ciências da Universidade de Lisboa
jorgemcgomes@gmail.com, {bc, apc}@di.fc.ul.pt

Resumo. Para os museus e galerias de arte a divulgação das suas exposições através da Web é importante, quer para atrair visitantes, quer para a difusão de património artístico e cultural. O desenvolvimento de ferramentas para a criação de exposições virtuais, além de possibilitar a divulgação das exposições, permite ainda auxiliar a concepção e montagem da própria exposição. Neste artigo apresenta-se uma aplicação que, a partir da modelação tridimensional, em formato X3D, do espaço físico do museu e da informação sobre as obras de arte a expor, permite a construção interactiva de uma exposição.

Abstract. Sharing virtual exhibitions through the Web is an important issue to museums and art galleries to spread their collection and to attract visitors. Software applications for building virtual exhibitions, besides providing a mean to spread exhibitions, may also be auxiliary tools to help a museum curator and his staff to conceive and mount an exhibition. This paper presents a software tool that allows users to interactively create a virtual exhibition, given an X3D file, with the 3D model of the physical environment of the museum, and information about the artworks.

Palavras-chave: Museus Virtuais, Aplicações Interactivas, X3D, Xj3D.

1 Introdução

A divulgação do acervo de museus e de galerias de arte através da Web é de grande relevância para as instituições. Além do contributo cultural em termos da divulgação de obras de arte, atingindo um público mais alargado do que aquele que habitualmente frequenta as suas instalações, a disseminação do acervo de um museu é também um meio para captar potenciais visitantes e um auxiliar para que estes possam tirar maior partido da sua visita. Por outro lado, é uma forma de dar visibilidade a obras que temporariamente não estão expostas, seja devido a restrições de espaço, seja pela sua fragilidade, seja por motivo de acções de restauro em curso. Para lá da colecção do próprio museu, através da Web podem ainda ser divulgadas exposições temporárias, que mostram a dinâmica cultural do museu.

A abordagem mais comum para a apresentação da colecção de um museu é através de páginas HTML com fotografias, texto e ligações para outras páginas. Outra

alternativa é a utilização de fotografias panorâmicas, como as que podem ser construídas em QuickTimeVR [1]. Estas fotografias panorâmicas permitem ao utilizador inspeccionar o espaço à sua volta com uma amplitude de 360º. Contudo, apesar de contribuírem para uma maior imersão no espaço expositivo, não permitem uma navegação livre nesse espaço. A passagem para outros locais do museu pode ser feita por selecção interactiva através de um mapa 2D com a marcação dos possíveis pontos de observação [2, 3]. Para atingir um maior grau de imersividade, com navegação livre no espaço tridimensional, a solução passa pela criação de modelos tridimensionais [4, 5]. Partindo de um modelo virtual do edifício, a criação da exposição deverá ser feita pela equipa do museu que, na maior parte dos casos, não inclui especialistas em informática. Por este motivo, uma ferramenta interactiva para criação de exposições, que permita a colocação de objectos de arte em locais seleccionados, auxiliará a equipa de um museu na fase de concepção e na fase inicial da montagem de uma nova exposição, bem como, a gerar uma exposição virtual a disponibilizar na Web.

Em [6,7] apresentou-se uma ferramenta concebida para a criação interactiva de exposições virtuais baseada em tecnologia Web3D [8], gerando ambientes tridimensionais em X3D. Entre as limitações detectadas nesta ferramenta destacam-se: o tratamento de apenas algumas das representações possíveis no formato X3D, a impossibilidade de alterar uma exposição criada numa edição anterior e a colocação apenas de obras de arte bidimensionais, como quadros e tapeçarias. Para colmatar estas limitações foi desenvolvida uma nova aplicação, mais genérica e com mais funcionalidades que se apresenta neste artigo. De entre estas novas capacidades destacam-se: o processamento do modelo tridimensional do cenário de modo a converter a sua descrição numa geometria que permita um tratamento uniforme de qualquer cenário; a possibilidade de edição de exposições já construídas; a criação de filtros auxiliares para identificação de superfícies elegíveis para colocação de obras de arte; a extensão do tipo de obras de arte que podem ser colocadas no espaço expositivo, nomeadamente, a possibilidade de inclusão de objectos 3D; a inserção de objectos auxiliares para suporte à apresentação de obras de arte, por exemplo, divisórias amovíveis e bases para colocação de esculturas.

Em seguida, na secção 2 referem-se trabalhos desenvolvidos no âmbito da divulgação cultural com recurso a modelação tridimensional. Na secção 3, descreve-se aplicação desenvolvida. Por último, na secção 4 apresentam-se as conclusões e o trabalho futuro.

2 Modelos Virtuais

Em várias aplicações desenvolvidas no âmbito da divulgação cultural, quer relativas ao acervo de museus e galerias de arte, quer de património histórico, têm sido utilizados modelos tridimensionais gerados em computador. No caso de património histórico desaparecido, a reconstrução virtual é por vezes a única forma de visualizar as alterações arquitectónicas sofridas por um edifício ao longo do tempo [9], [10], ou o seu aspecto antes da sua ruína causada pela acção do tempo, por catástrofes naturais ou por guerras [11, 12]. Entre outras referências nesta área, as que são indicadas têm

em comum a utilização de modelos tridimensionais em VRML ou X3D. Esta tecnologia serve também de base à criação de modelos virtuais de museus, uma vez que neste contexto, os ambientes virtuais desenvolvidos se destinam, na maior parte dos casos, a ser visualizados através da Web.

Tentando responder à necessidade de criar ferramentas acessíveis para utilizadores sem formação específica em informática, como acontece normalmente com as equipas de museus, têm sido propostas algumas ferramentas para auxílio à construção de exposições virtuais. Em [13] descreve-se um editor gráfico 2D que permite, por um lado, a edição da planta do espaço de exposição, alterando, por exemplo, a cor ou textura associada a uma dada parede, e por outro lado, a colocação de quadros nas paredes. Esta segunda tarefa é realizada usando duas janelas auxiliares: uma com a visualização das áreas cobertas pelos quadros colocados, outra com a interface para escolha da obra e indicação das coordenadas para colocação do quadro. Não é tratada a inserção interactiva de objectos tridimensionais.

No âmbito do projecto ARCO (Augmented Representation of Cultural Objects) desenvolveram-se um conjunto de ferramentas destinadas a construir e gerir exposições virtuais de museus [14]. Neste conjunto incluem-se ferramentas para o apoio: à produção de modelos digitais de obras de arte, à gestão de um repositório, onde podem ser guardados vários tipos de representações para estas obras, e à sua visualização num espaço tridimensional. A criação de exposições virtuais pode ser feita através de um conjunto de "templates" seleccionando os parâmetros adequados e os modelos VRML/X3D guardados no repositório. Estes "templates", que definem tanto o aspecto visual como o comportamento da apresentação [15], são construídos em X-VRML, uma linguagem de alto nível baseada em XML. A criação de "templates" permite maior versatilidade na apresentação da exposição, mas requer alguns conhecimentos de informática.

Um exemplo mais recente de criação de um museu virtual é descrito em [16]. O trabalho desenvolvido teve por objectivo a divulgação, através da Web, do Museu de Arte Contemporânea da Macedónia, em Tessalónica, na Grécia. As obras de arte digitalizadas e o modelo do espaço físico do museu são convertidos num formato Web3D (VRML/X3D) para serem visualizados através da Web. Não é reportada a construção de uma aplicação própria para ser utilizada pela equipa do museu, mas, em contrapartida, uma das funcionalidades criada para os visitantes virtuais é a construção interactiva de uma galeria de arte. Os objectos de arte a visualizar podem ser seleccionados através de pesquisas à base de dados de obras de arte, baseadas em palavras chave, como o nome do autor ou o nome da obra, entre outros, e são depois colocados por "arrastamento" no espaço virtual. Não são indicados detalhes sobre esta aplicação.

Como já foi mencionado, em trabalho anterior [6, 7], desenvolvido em colaboração com o Instituto Açoriano de Cultura, foi criada uma aplicação para apoio à criação de exposições virtuais pelo conservador de um museu e pela sua equipa. Consideraram-se como requisitos para esta aplicação a sua utilização por pessoas sem especialização em informática e a possibilidade de reutilização para vários espaços de exposição. Optou-se por suportar modelos virtuais em X3D. A manipulação da cena é feita com recurso ao Xj3D e a informação sobre os quadros está guardada numa base de dados MySQL. Decompôs-se a construção da exposição virtual em duas fases: na primeira fase identificam-se as superfícies onde podem ser colocados quadros,

criando um novo ficheiro com sensores de toque associados a estas superfícies; na segunda fase constrói-se a exposição virtual pesquisando quadros na base de dados e seleccionando a superfície onde devem ser colocados. Para ultrapassar as limitações identificadas na ferramenta desenvolvida, foi concebida uma nova aplicação com as características já referidas na introdução e que são explicadas mais detalhadamente na secção seguinte.

3 Ferramenta para Criação Interactiva de Exposições Virtuais

Como já foi referido, no desenvolvimento desta ferramenta foi dado ênfase à criação de mecanismos para a concretização de uma exposição virtual de forma interactiva e acessível a pessoas não especialistas em informática. Além disso, pretende-se que possa ser usada para diferentes espaços físicos.

Uma vez que a montagem da exposição consiste na selecção de obras de arte e associação de cada uma delas à superfície onde vai ser exposta, um dos problemas que se colocam para o tratamento genérico de qualquer espaço expositivo, é a necessidade de juntar à descrição deste espaço, de forma automática, a capacidade de escolha e selecção destas superfícies. No caso do X3D este processo passa pela adição de sensores de toque às superfícies elegíveis para colocação de objectos. Mais ainda, para ser possível fazer ajustamentos a uma exposição já criada é necessário guardar informação sobre a estrutura da cena e as alterações nela introduzidas. De modo a tratar estas situações, foi necessário desenvolver um processo de conversão da cena inicial numa estrutura que permita a sua manipulação e tratamento adequado.

Outro aspecto contemplado foi a criação de um mecanismo uniforme para a colocação de objectos na cena independente do objecto concreto, de forma a estender o tipo de objectos que podem ser colocados na cena.

Em seguida apresenta-se a arquitectura da aplicação, descrevem-se as adaptações e funcionalidades introduzidas, bem como a interface com o utilizador.

3.1 Arquitectura da Aplicação

A arquitectura da aplicação baseia-se num desenho modular de forma a poder facilmente estender as funcionalidades suportadas, ponto que é essencial neste problema uma vez que para a construção de uma exposição virtual pode ser necessária uma grande diversidade de funcionalidades e objectos. O funcionamento é repartido por modos de edição, cada um deles implementado por um módulo independente.

Por modo de edição entenda-se um modo capaz de interagir com a cena para a modificar de alguma forma específica e dotado das capacidades de ser activado, desactivado, gravado o seu estado lógico e carregado posteriormente para reedição da exposição. Além disso, tem a capacidade de adicionar as alterações, efectuadas à descrição inicial da cena, no ficheiro X3D que guarda o resultado final da exposição construída. Em cada momento, apenas um modo de edição pode estar activo, e a qualquer momento pode mudar-se de um modo para qualquer outro.

No entanto, existem processos que são comuns a vários módulos, por exemplo, a colocação de objectos na cena. Para suportar facilmente a criação de módulos que se

baseiem em colocação de objectos, é definido no núcleo central (Core) da aplicação um módulo abstracto que define todas as funcionalidades comuns a estes módulos, como a colocação e movimentação dos objectos e o funcionamento geral do modo de edição. Cada módulo concreto estende este módulo abstracto, e fica encarregado de definir as especificidades do mesmo, nomeadamente, as características do objecto concreto que é colocado, funcionalidades que permitam a escolha do mesmo, e eventualmente alterações ao funcionamento normal do modo de edição. Foram concretizados três módulos para a colocação de: objectos de arte bidimensionais, objectos de arte tridimensionais e estruturas de apoio à exposição, como divisórias amovíveis e bases para colocação de esculturas.

No núcleo central (Core) são também definidas as funcionalidades relacionadas com a gravação e carregamento do estado da aplicação e exportação da cena final (Project Management), bem como o módulo de eleição e detecção de superfícies, ao qual os restantes módulos têm acesso (Surface Module).

Na Fig. 1 apresentam-se os vários módulos que compõem a aplicação, bem como a articulação entre eles.

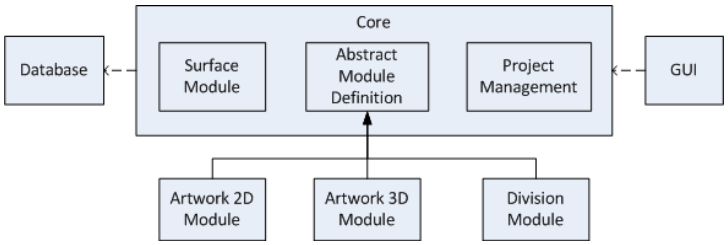


Fig. 1. Módulos da aplicação

A informação relativa às obras de arte é guardada numa base de dados integrada na aplicação através do SQLite. Optou-se por uma base de dados integrada pois dispensa a instalação e configuração de servidores e bases de dados. Apesar de ser mais interessante a ligação com bases de dados já existentes, esta solução levantaria certamente problemas de compatibilidade. A informação sobre os objectos é colocada na base de dados através de uma ferramenta oferecida com a aplicação (SQLiteStudio). Em alternativa pode ser construída uma interface gráfica na aplicação que permita, de forma *user-friendly* e orientada ao domínio, a inserção desta informação.

A aplicação é construída sobre Java SE 1.6, e para trabalhar com o X3D utiliza principalmente a API Xj3D, versão 2.0M1, que constrói e manipula o grafo lógico da cena. Para acesso e modificação dos conteúdos deste grafo são utilizados os métodos do SAI (Scene Access Interface), também parte integrante da especificação X3D. Por conveniência recorreu-se à API Java3D 1.3 para algumas fases do processamento geométrico. A interface gráfica (GUI) é construída sobre Java Swing.

3.2 Uniformização da Geometria da Cena

Apesar de o X3D ser um formato bem definido, permite que haja múltiplas representações internas para obter o mesmo resultado visível, devido à grande variedade de nós que existem para descrever a geometria de um objecto. Tal variedade permite, por exemplo, que os programas de modelação tridimensional que exportam ficheiros para o formato X3D utilizem processos de conversão diferentes, recorrendo a técnicas e a nós distintos. Assim, analisar uma cena X3D, da qual se desconhece o processo de criação, levanta alguns problemas que devem ser resolvidos de forma a poder interpretar de forma fácil, completa e correcta a sua geometria.

Uma vez que a aplicação precisa de interpretar a geometria da cena e de alterar alguns dos seus aspectos, por exemplo adicionar sensores, tornou-se evidente que era necessário criar uma camada que abstraísse a geometria original da cena. Para este efeito concebeu-se um processo que não altera a descrição inicial da cena e junta uma nova definição da sua geometria, que contém a informação necessária para adicionar novos objectos à cena. Esta camada de abstracção da geometria inicial da cena é composta por um conjunto de superfícies, sendo cada superfície uma área plana caracterizada por uma só normal e constituída por um número arbitrário de triângulos adjacentes que a definem. Por exemplo, no modelo de um edifício, uma destas superfícies corresponderá a uma parede ou ao chão.

O processo de conversão da descrição inicial da cena num conjunto de superfícies integra os seguintes passos: conversão de toda a geometria para triângulos; agregação dos triângulos em superfícies; eliminação de elementos geométricos repetidos e identificação de duas faces para cada superfície.

Para converter toda a geometria para triângulos, utilizam-se os filtros fornecidos pela ferramenta de conversão distribuída juntamente com o Xj3D. Posteriormente, a agregação de triângulos junta em superfícies os triângulos adjacentes com normais paralelas. Uma vez que a geração do ficheiro inicial X3D, através da exportação de modelos criados com ferramentas de modelação, pode incluir repetição de geometrias, é necessário proceder nesta fase à eliminação das repetições. Finalmente, atendendo a que cada superfície pode ser observada segundo as suas duas faces e que é necessário considerar um sensor de toque para cada uma delas, é preciso proceder à identificação das duas faces de cada superfície.

3.3 Selecção de Superfícies

Um aspecto importante na construção de uma exposição é a selecção das superfícies elegíveis para colocação de obras de arte, uma vez que nem todas são adequadas para este efeito.

Assim, antes de iniciar a colocação de objectos de arte, o utilizador deve escolher quais as superfícies apropriadas para a sua exibição. Esta selecção pode ser interactiva, escolhendo uma a uma todas as superfícies pretendidas, ou recorrendo a filtros configuráveis. Estes filtros analisam cada superfície e decidem se esta deve ser eleita ou não. Até agora foram definidos os seguintes filtros:

- Filtro de área: aceita as superfícies que tenham uma dada área mínima.

-Filtro de normal: aceita as superfícies cuja normal verifica as condições impostas pelo utilizador.

Estes filtros podem ser combinados e é possível activá-los também na negativa, ou seja, é possível rejeitar superfícies com base nos filtros.

3.4 Colocação de Objectos

De modo a permitir estender o tipo de objectos que se podem juntar à cena, definiu-se um conjunto de parâmetros básicos que caracterizam um objecto: a sua caixa envolvente, a base de contacto com a superfície onde é aplicado, a normal a esta superfície, correspondente à face ou ao volume visível do objecto, e a orientação do seu bordo superior. Para simplificar o processamento, assume-se que os objectos a colocar na cena têm, eventualmente por aplicação de transformações prévias, os seguintes valores para estes parâmetros: a superfície de apoio é paralela ao plano XZ, a normal à superfície é um vector com a direcção e o sentido do semi-eixo positivo dos YY e a orientação da linha de topo corresponde ao semi-eixo positivo dos XX. Em função destes parâmetros, é possível, por aplicação de transformações geométricas, colocar qualquer objecto sobre uma superfície.

Após seleccionar o objecto a colocar na cena, escolhe-se com o rato a superfície da área de representação onde este vai ser aplicado. O centro da superfície de contacto do objecto é colocado sobre o ponto da superfície seleccionado com o cursor. Posteriormente, a posição do objecto pode ser ajustada através de translações ao longo da superfície e de rotações. Estas transformações são controladas através de botões da interface. O objecto pode, alternativamente, ser recolocado noutra superfície.

Como já foi referido, para cada tipo de objectos existe um módulo específico que tem em atenção as suas especificidades. No caso de obras de arte bidimensionais, estas são guardadas como imagens na base de dados, juntamente com outras informações relativas à obra. Quando são colocadas na cena é criado um paralelepípedo e a imagem é aplicada como textura numa das faces. A base de dados também contém as dimensões físicas da obra, de modo a que possa ser colocada na exposição com a dimensão correcta.

As obras de arte tridimensionais são guardadas como modelos X3D e a sua orientação no espaço deve cumprir as regras acima descritas. A sua caixa envolvente é calculada automaticamente por análise da sua geometria. Para este efeito, o modelo inicial é transformado numa descrição que apenas contenha nós com as coordenadas explícitas e em seguida é feita uma pesquisa nestas coordenadas para obter os extremos da caixa envolvente. Quando o objecto tridimensional é inserido na cena, os nós X3D que o descrevem são adicionados à cena já existente.

Com já foi referido, além de obras de arte, é possível juntar à cena objectos para suporte à sua apresentação, como divisórias amovíveis ou bases para a colocação de objectos (plintos). Para este efeito é possível colocar na cena objectos paralelepipédicos com dimensão e cor escolhidas pelo utilizador. Uma vez colocados na cena, as superfícies que os constituem são consideradas também superfícies elegíveis para colocação de obras de arte e podem ser escolhidas no modo de selecção de superfícies. Assim, é possível colocar outros objectos sobre estas divisórias.

3.5 Geração do Ficheiro X3D com a Exposição

O ficheiro X3D com a descrição da exposição desenhada é construído recorrendo a um processo de exportação, uma vez que não é possível através do Xj3D exportar o grafo de uma cena para um ficheiro X3D. A exportação é feita modificando o ficheiro da cena original, que é manipulado através da interface DOM (Document Object Model) fornecida pela API do Java para processamento de XML.

De modo a manter também a modularidade neste processo, cada módulo é responsável por alterar independentemente o documento XML de forma a reflectir as mudanças realizadas no contexto do mesmo.

A Fig. 2 mostra a visualização, no browser do Xj3D, do ficheiro resultante da criação de uma exposição.

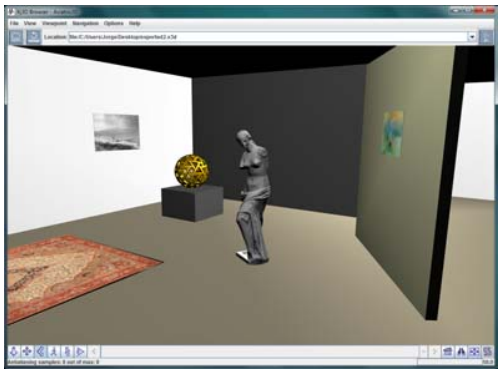


Fig. 2. Visualização do resultado da construção de uma exposição

3.6 Mecanismo de Reedição de Exposições

A reedição de uma exposição não pode simplesmente ser feita a partir do ficheiro X3D resultante da sua construção: é necessário conhecer as alterações introduzidas à cena inicial.

Para permitir a reedição de exposições já construídas, concebeu-se um mecanismo baseado num conjunto de estruturas de dados auxiliares que guardam informação sobre todos os objectos que se juntam à cena. É possível guardar num ficheiro, designado por ficheiro de estado, o conteúdo destas estruturas. Um ficheiro de estado reflecte o estado corrente de edição de uma cena. Posteriormente, para voltar a editar uma exposição, repõe-se o conteúdo das estruturas de dados auxiliares a partir do ficheiro de estado relativo a essa exposição. A partir da cena inicial e do conteúdo destas estruturas de dados reconstitui-se a cena correspondente ao estado de edição gravado.

3.7 Interface

A interface com o utilizador está dividida em cinco zonas (Fig. 3): a barra de menus com os comandos usuais, como a selecção e a gravação de ficheiros; a zona de selecção do modo de edição, sob a barra de menus; a área de representação onde é desenhada a cena tridimensional; a barra com botões de navegação, sobre a área de representação; e a zona de opções específicas de cada modo de edição, à esquerda da área de representação.

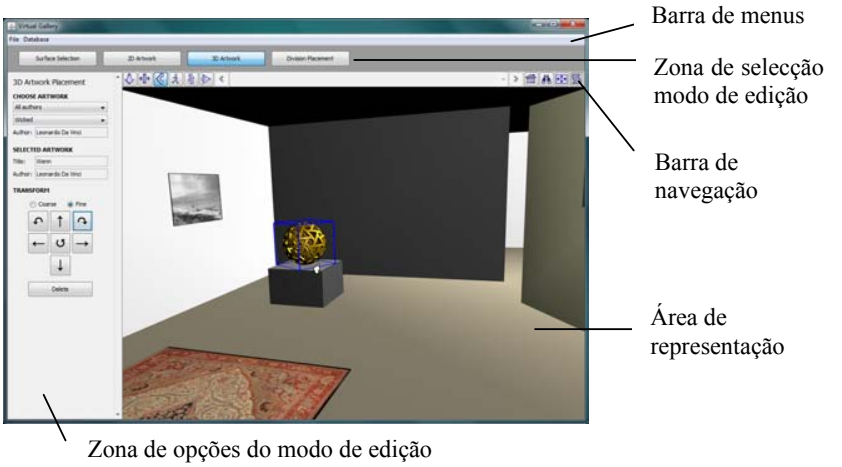


Fig. 3. Interface para colocação de objectos tridimensionais

Os modos de edição que contemplam a colocação de objectos têm em comum o seguinte:

- Na sua zona de opções específicas existe um conjunto de botões para a aplicação de translações e rotações a um objecto seleccionado.
- As superfícies eleitas para colocação de objectos ficam activas para selecção.
- Os objectos colocados na cena, do tipo correspondente ao modo de edição activo, são susceptíveis de ser seleccionados.

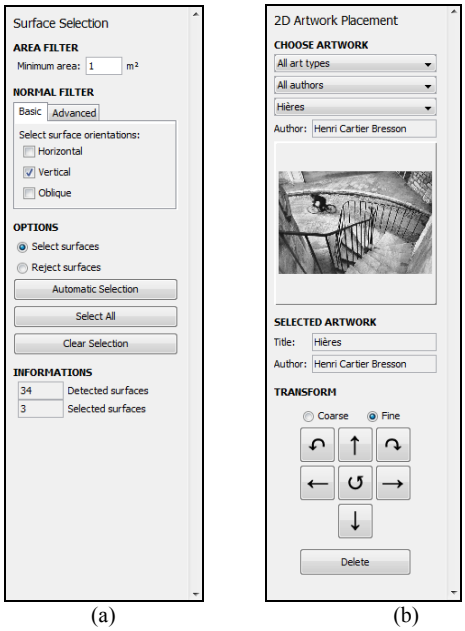
Apresentam-se em seguida os aspectos particulares da interface de cada modo de edição.

Selecção de Superfícies. Neste modo de edição é disponibilizada uma interface simples para configurar os filtros de selecção anteriormente descritos (Fig. 4 (a)). Quando uma superfície está seleccionada a sua cor toma tons mais avermelhados para se destacar (Fig. 5).

Colocação de Objectos de Arte Bidimensionais. A activação deste modo suporta a colocação de objectos de arte a duas dimensões nas superfícies. A zona de opções deste modo de edição contém os objectos de interface para a pesquisa na base de dados de obras de arte (Fig. 4 (b)). É mostrado um *thumbnail* da obra escolhida, quer por pesquisa na base de dados, quer por selecção com o cursor de uma obra já colocada na cena.

Colocação de Objectos de Arte Tridimensionais. A interface deste modo (Fig.6 (a)) é semelhante à de colocação de objectos bidimensionais, contudo não é mostrada uma visualização rápida do modelo da obra seleccionada na base de dados em benefício do desempenho da aplicação.

Colocação de Novas Estruturas de Apoio. Neste modo de edição são fornecidas opções básicas ao utilizador para escolher as dimensões do paralelepípedo e seleccionar a sua cor. A interface para escolha de cor corresponde ao Color Chooser disponibilizado pela API do Java (Fig.6 (b)).



(a) (b)

Fig. 4. Zona de opções para: (a) selecção de superfícies; (b) colocação de objectos 2D

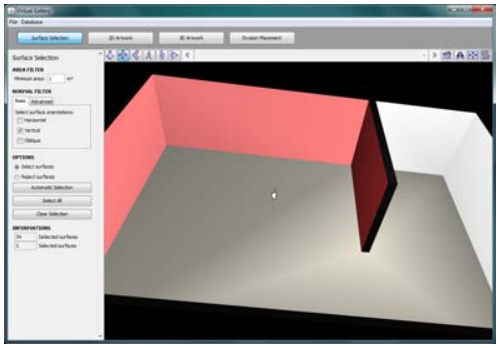
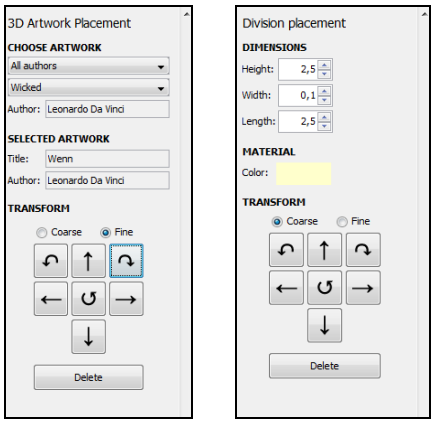


Fig. 5. Modo de selecção de superfícies com as paredes seleccionadas a vermelho



(a) (b)

Fig. 6. Zona de opções para: (a) selecção de objectos 3D; (b) colocação de divisórias

4 Conclusões e Trabalho Futuro

Apresentou-se neste artigo uma ferramenta para construção interactiva de exposições virtuais que se destina a preparar a montagem de exposições por equipas de museus e cujo resultado final pode ser divulgado através da Web. Tendo por base a experiência adquirida em trabalho anterior e a necessidade de ultrapassar as suas limitações, foram criados mecanismos que permitem a leitura de qualquer cenário descrito em X3D, a reedição de exposições já montadas, a extensão do tipo de obras de arte que podem ser exibidas e a inclusão de estruturas de suporte à exibição de obras de arte.

A estrutura de desenvolvimento modular permite a adição de novas funcionalidades. No seguimento do trabalho desenvolvido está planeada a implementação de um módulo para a colocação de fontes de luz na exposição, através de projectores colocados em superfícies ou em calhas próprias para eles. Além da selecção da posição e orientação, deverá ser possível escolher a intensidade, cor e tipo destas fontes de luz.

Outros módulos a desenvolver prendem-se com funcionalidades relativas: à colocação de objectos suspensos num ponto de aplicação; à pintura de superfícies; à colocação de molduras em quadros; à projecção de vídeos em superfícies; à reprodução de música ambiente em função do local que está a ser visualizado no momento; à definição de *viewpoints* para criação de percursos pré-definidos.

A evolução futura desta ferramenta depende também do resultado de testes de avaliação a realizar por especialistas na área.

Agradecimentos. Este trabalho foi financiado por uma bolsa da Universidade de Lisboa/Fundação Amadeu Dias.

5 Referências

1. QuicktimeVR, http://www.apple.com/br/quicktime/technologies/qtvr/ (9-7-2010)
2. Museu do Louvre, Paris http://musee.louvre.fr/visite-louvre/index.html?defaultView= entresol.s489.p01 &lang =ENG (9-7-2010)
3. National Gallery of Art, Washington, http://www.nga.gov/exhibitions/calder/realsp/roomenter-foyer.htm (9-7-2010)
4. Baylyl Art Museum, University of Virgínia, http://www2.lib.virginia.edu/artsandmedia/artmuseum/docs/virtual.html (9-7-2010)
5. Reprodução de uma galeria do Victoria and Albert Museum, Londres, http://www.arco-web.org/Virtual/dresses.php (9-7-2010)
6. Semião , P. M., Carmo, M. B.: Galeria de Arte Virtual, Actas do 15º Encontro Português de Computação Gráfica, poster. (2007)
7. Semião , P. M., Carmo, M. B.: Virtual Art Gallery Tool, Proceedings GRAPP 2008, pp 471-476, (2008)
8. Web 3D, http://www.web3d.org/ (9-7-2010)
9. El-Hakim, S., MacDonald, G., Lapointe, J.-F., Gonzo, L., Jemtrud, M.: "On the Digital Reconstruction and Interactive Presentation of Heritage Sites through Time", Proceedings VAST'06. (2006)
10. Hetherington, R., Farrimond, B., Presland, S.: Information rich temporal virtual models using X3D. *Computers & Graphics*, Vol 30 (2), pp. 287-298. (2006)
11. Vlahakis, V., Ioannidis, N., Karigiannis, J., Tsotros, M., Gounaris, M.: Virtual Reality and Information Technology for Archaeological Site Promotion, Proceedings BIS'02. (2002)
12. Ramic-Brkic, B., Karkin, Z., Sadzak, A., Selimovic, D., Rizvic, S.: Augmented Real-Time Virtual Environment of the Church of the Holy Trinity in Mostar, VAST'09, pp141-148. (2009)
13. Hrk, S.: Virtual Art Gallery, CESCG 2001. (2001)
14. Wojciechowski, R., Walczack, K, White, M., Cellary, W.: Building Virtual and Augmented Reality Museum Exhibitions, Proceedings of the 9th International Conference on 3D Web Tecnology, pp 135-144. (2004)
15. Walczak, K., Cellary, W., White, M., Virtual Museum Exhibitions, IEEE Computer, Vol 39 (3), pp 93-95. (2006)
16. Patias, P., Chrysantou, Y., Sylaiou, S., Georgiadis, Ch., Michail, D. M., Stylianidis, S.: The Development of an E-Museum for Contemporary Arts, Conference on Virtual Systems and Multimedia. (2008)

GUItar and FAgoo: Graphical interface for automata visualization, editing, and interaction*

André Almeida Nelma Moreira Rogério Reis
{bernarduh,nam,rvr}@ncc.up.pt

DCC-FC & LIACC, Universidade do Porto
R. do Campo Alegre 1021/1055, 4169-007 Porto, Portugal

Abstract. GUItar is a graphical environment for graph visualization, editing, and interaction, that specially focuses in finite automata diagrams. The application incorporates mechanisms to facilitate the editing of these graphs. It also provides a style manager that allows the creation of rich state and arc styles to be used in the drawing of its objects. This style manager allows the system to cope with complex styles, broaden the application scope to graphical representations of other computational models like transducers or Turing machines. GUItar also has a foreign function call (FFC) mechanism for the easy integration of external modules and libraries like automata symbolic manipulators or graph drawing libraries. For automatic graph drawing we are developing FAgoo, a package that seeks to provide tools capable of finding pleasant graph drawings. FAgoo implements graph drawing algorithms that find embeddings which the user, with minimal manual changes, can adjust to its aesthetically taste. Both GUItar and FAgoo are on going projects licensed under GPL.

1 Introduction

GUItar [1] is a graphical environment tool for finite automata visualization and editing. This application incorporates mechanisms, like the auto adjustment of the nodes to avoid overlaps and the automatic positioning of the arcs which assist the user through the graph drawing and visualization. GUItar also provides powerful styling tools that not only allow the editing of node and arc styles but also allows the creation of new node structures. Furthermore we present the foreign function call (FFC) mechanism which is used to access external modules or libraries as FAdo and FAgoo [1].

FAdo is a tool for symbolic manipulation of formal languages and specially finite automata that can be incorporated with GUItar. Since most FAdo manipulations result in finite automata diagrams with no embedding, we are developing FAgoo which is a graph drawing library that specially focuses in that type of diagrams. Finite automata diagrams require additional aesthetic and graphical

* This work was partially funded by Fundação para a Ciência e Tecnologia (FCT) and Program POSI, and by project ASA (PTDC/MAT/65481/2006).

constraints over other type of graphs. Finite automata diagrams, for example, are normally read from the left to the right therefore initial states are placed on the left and final states more to the right. FAgoo is a Python module written in C allowing us to maintain good performance and at the same time provide a high-level interface. In this paper we will describe the main components of GUItar as well as some algorithms already implemented in FAgoo.

2 Graph Drawing Libraries and Applications

There are several graph drawing libraries available with many layout algorithms for generic and specific types of graphs. Most of these libraries focus in a specific type of graphs in order to achieve better drawings. Restricting the type of graphs that an algorithm have to deal with, results in having graphs with particular properties which usually facilitates their drawings. Although there are many applications and libraries as aiSee [2], yED from yWorks [3], Open Graph Drawing Framework (OGDF) [4] and Graphviz [5] for automatic graph drawing, the algorithms implemented by these software do not fit the drawing conventions of finite automata drawings. This is because they were not specially designed to deal with finite automata drawings. JFLAP [6] is an application that aims to provide a way to experiment with formal languages representations, in particularly finite automata. Clearly JFLAP do not focus its work on the visualization and layout of the finite automata, thus, the available layout algorithms are very basic and simple.

3 GUItar

GUItar is an ongoing project which aims to provide a software tool for finite automata visualization and editing. Although GUItar specially focuses in finite automata diagrams, it supports other types of diagrams. Currently GUItar is implemented in Python and uses wxPython [7] graphical toolkit. The graphical interface basic frame is composed by a menu bar, a tool bar and a notebook. The menu bar is dynamically built from XML [8] configuration files. The notebook can handle multiple pages, each one containing a canvas. The canvas is implemented using the wxPython's FloatCanvas module which provides a set of graphical objects that can be bound with mouse events. To be able to draw labeled arcs, a new object called ArrowSpline had to be created.

3.1 Styles

In order to have a good platform for graph visualization and editing, we need to cope with a wide range of node and arc styles. GUItar provides a node and arc style manager that not only allows management of multiple styles, but also provides interactive creation and editing of complex node structures. The graphical representation of a node on GUItar consists in a set of objects from ellipses,

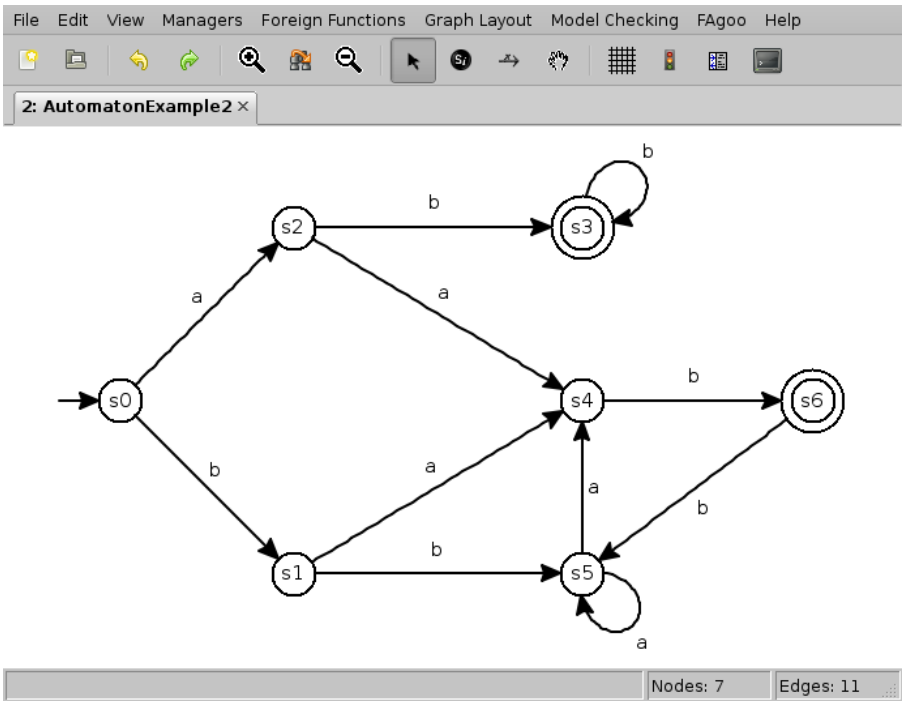


Fig. 1. An automaton created in GUItar.

rectangles, arrow splines and scaled texts. There must exist at least an ellipse or a rectangle to ensure that the node has a place to dock the incoming and outgoing arcs. It must also have one scaled text to place the node's label. This node structure allows the creation of complex nodes, enriching the graph visualization. For example, in finite automata diagrams we can represent final states by using two concentric ellipses, or initial states using a arrow and a ellipse. The Fig. 2 shows the node style manager of GUItar, editing a style that could be used to represent a state which is initial and final.

The node and arc labels can be either simple or compound. Simple labels are just text strings, while compound labels have custom fields with values specified by the user. The user can choose either to display or not each label field, and in this way, extra data can be associated to nodes and arcs.

These features can accommodate many purposes, expanding GUItar's scope to a larger range of graph types such as transducer diagrams, Turing machines, and others.

3.2 Visualization

Large graphs are difficult to visualize. If we are focusing in a part of a graph and we want to abstract ourselves from the rest of the graph we can select that part

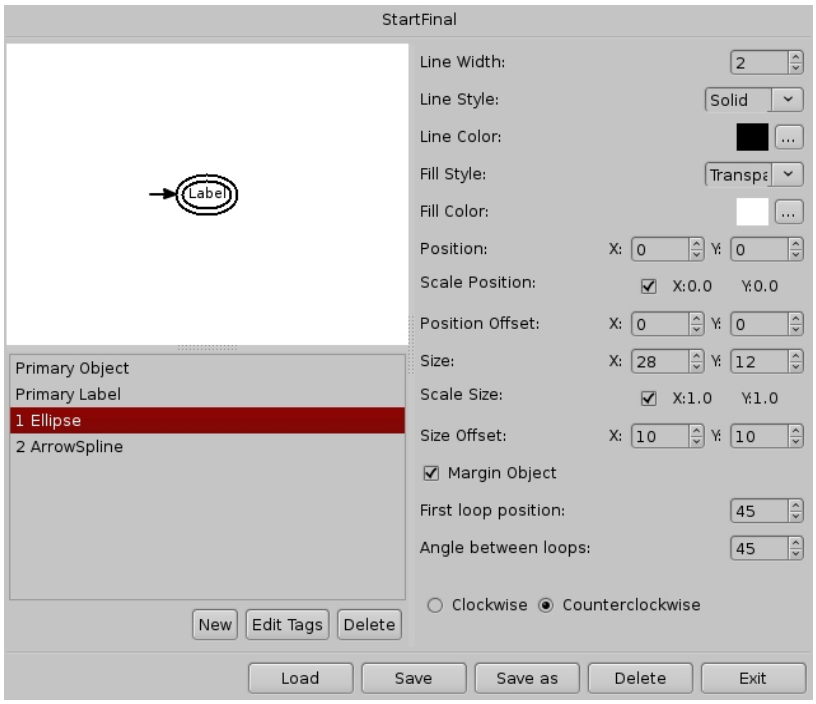


Fig. 2. GUItar's node style manager.

of the graph and ask the application to find a specific embedding that favours its visualization. There is also the case where we may want to have a overview and simplify some parts of the graph collapsing a subgraph into a node. One solution is to replace a subgraph by a node and transforming all the external arcs of the subgraph in the respective arcs to the node.

3.3 Graph Manipulation

In GUItar it is possible to collapse multiple arcs between two nodes. To collapse multiple arcs their labels must be merged. By default the concatenation operation is used, but other operations can be defined. As for the resulting style, if all arcs share the same style then that one is used. If different styles are present, one is arbitrarily chosen or the user is prompted to do it.

Two or more nodes can be merged into one. To do this there are three aspects to take in consideration: the labels, the styles, and the arcs. The labels' merging and the style selection are done as above. The arcs of the merging nodes are replaced by arcs to the resulting node, which can lead to the creation of multiple arcs. Further collapsing of these resulting arcs can then take place.

3.4 FFCs

We do not intend this project to be a new monolithic graph visualization and editing tool, but we see it more as a hub where graph manipulation libraries can, together, provide better visualization and manipulation tools. This is achieved by a FFC mechanism, using a Python interface to access the external tools (see Fig. 3). There are three types of FFC: module FFC, object FFC and interactive FFC. In the first case, the FFC calls a function directly from an external Python module. In the second case, it creates a foreign object and then calls methods of that object. In the last case, when a specified event occurs, the FFC triggers the respective handler function from an external Python module that will return a sequence of actions as script commands. FFCs require an XML configuration file that specifies the available methods. FFCs can create their own menu entries which makes its integration in GUItar smooth and practical. Most of the GUItar tools are implemented using FFCs that interface FAdo and FAgoo.

3.5 Animations

Animation can be a good way to illustrate a complex algorithm behavior. One application of interactive FFCs is algorithm animation. A simple algorithm as the one that finds a path between two nodes can be annotated with commands and events to animate its execution. These annotations allow to control the canvas behavior and its contents. To control the animation flow, GUItar can provide a set of interactive controls or the FFC can provide its own external interface. An example of a more complex animation is the deterministic finite automata minimization that uses nodes merging to illustrate some of the transformations during the algorithm execution.

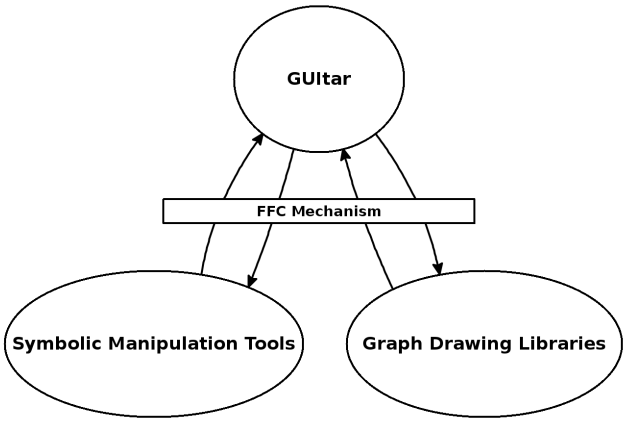


Fig. 3. A FFC mechanism overview.

3.6 Graph Classification

The GUItar classification mechanism allows to test if a graph belongs to a certain class by checking if the graph verifies a set of properties. These properties can test graphical properties (e.g. if arcs have arrows) or semantic properties (e.g. if a finite automaton is deterministic). A few of these methods are predefined in GUItar to check the most usual graphical properties of a graph. Access to external libraries with FFCs can be used to test graph properties, broaden the class range. Biconnectivity and planarity tests can be done, for example, using FAgoo.

A friendly interface is available for graph classification (see Fig. 4). This interface lists the graph properties and identifies the ones that are verified for the current graph. The user can create his own classes by stating the properties that the class must comply. It is also possible to export and import these class definitions.

Fig. 4. GUItar's interface for graph classification.

3.7 Semaphores

When editing a graph it can be useful to constraint the actions performed such that the resulting graph does not leave a certain class. The GUItar Semaphore tool assists this task by warning the user, or even restricting his actions. For example, suppose that we have a deterministic finite automata (DFA) as the result of some manipulation, and we want to edit it. We can enable the semaphore for DFAs to ensure that the changes that we apply to the graph do not compromise the DFA class definition.

New Semaphores can be created by extending the Semaphore base class and declaring them in a XML configuration file. An image of a traffic sign is associated to each semaphore which its light color represent the current state of the graph evaluation. There is also an image of a small padlock that when closed means that actions are restricted, i.e., do not allow actions that compromise the desired graph properties.

3.8 Import and Export

GUItar store its graphs using GUItarXML [9], which is an XML format specially designed for this application and based on GraphML [10]. GUItar also imports and exports to other formats, converting from and to GUItarXML. Currently the available exporting formats are GraphML, dot [11], Vaucanson-G [12] and FAdo. It is also possible to import from all these formats with the exception of Vaucanson-G. The Xport mechanism provides an easy way to add new export and import methods to GUItar. These methods can be coded in Python or use XSL transformations [13].

3.9 Object Library

Automata manipulation involves many operations with automata which result in large sets of new automata. What Object Library offers is a way of tracing these operations (methods) and all the objects involved. With this information it is possible to maintain an history of these operations and know the origins of each object, as well as recreate them from the original object. This information can be used to enrich an automata database by adding complementary information about the automata origins. Another feature is the possibility of create scripts with these operations, which the user can save and then apply to other objects. An interesting application for this tool is the creation of scripts with sequences of graph drawing algorithms, to generate specific layouts that then could be applied to several graphs.

4 FAgoo

Graph drawing is an active area of research with a lot of documented algorithms for generic and specific graph types. However, there are not many specifically designed for finite automata diagrams. To enhance the readability of each type of diagrams, they are normally drawn according to a set of conventional rules. A finite automata is better read if it flows from left to right. Initial states must, thus, be placed in the left and final states tend to be pushed to the right. The labels are another particularity of finite automata diagrams. Finite automata labels can be very complex and large. A single arc can have a label with several strings attached, each one being complex, like a regular expression. Another constraint is the arcs and its labels placement. Arcs from the left to the right are placed above the ones from the right to the left, with labels placed on their left side. These constraints benefit the readability of these diagrams. Finally the frequent occurrence of loops which is not so usual in another type graphs, is another characteristic of finite automata diagrams. Generic graph drawing algorithms usually discard loops during the layout process and arrange them in a final stage, but loops are frequent in finite automata diagrams and can have complex labels which hardens its positioning task.

4.1 Drawing Planar Graphs

When drawing a graph, edge crossing reduces its readability, thus, making this an important aspect to consider [14]. A planar graph can be drawn on a plane without edge crossing, in particular it can be drawn only using straight-line edges. The algorithm implemented for planarity test is the one presented by Hopcroft and Tarjan [15], which has a linear time execution and can be extended to either construct a planar embedding (if the graph is planar) or determine the Kuratowski subgraph (if the graph is non-planar) [16].

The implemented straight-line drawing algorithm assumes that the input graph is triangulated, i.e., every face has exactly three vertices. To triangulate a planar graph while minimizing the maximum degree, Kant presented a algorithm that is a good approximation of the optimal solution, but this algorithm takes as input a triconnected graph. Since FAgoo currently does not implements a triconectivity augmentation algorithm, the canonical triangulation algorithm presented by Kant [17] was implemented. This algorithm only requires the input graph to be biconnected and computes a canonical ordering while triangulating a planar graph. This algorithm was slightly modified to compute a left most canonical (lmc) ordering. This ordering is a generalization of the canonical ordering of de Fraysseix et al. [18] and it is needed for the straight-line drawing algorithm.

Most graph drawing algorithms require that the input graph to be biconnected, i.e., a connected graph that remains connected after the removal of any vertex. FAgoo implements algorithms to test a graph biconnectivity, that with a few modifications computes the graph biconnectivity tree (BC-Tree), and the biconnectivity augmentation. There are two types of nodes in a BC-Tree, the

B-Nodes and the C-Nodes. The B-Nodes represent the maximal biconnected subgraphs and the C-Nodes represent the cutvertices. There is an edge between a C-Node and a B-Node if that C-Node belongs to the biconnected component represented by the B-Node. The biconnectivity augmentation algorithm takes a planar embedding of each biconnected component of the graph and its BC-Tree to biconnect the graph while preserving its planarity. This is a simple linear time algorithm [17], which is an adaptation of the one presented by Read [19].

4.2 Drawing Non-Planar Graphs

Finding an embedding for a non-planar graph that minimizes its edge crossing is NP-hard [20]. One possible approach for non-planar graphs is to remove arcs from its Kuratowski subgraph, that can be found in linear time, until the graph is planar. Finding this minimal set of arcs is the hard task. Another approach for non-planar graphs is to find the maximum planar subgraph. Again, this problem is NP-hard [21]. These problems have been researched over the last 20 years and still do not have good solutions for generic graphs. As future work we intend to develop these approaches and implement them in FAgoo.

4.3 Subgraph Drawings

Some times it is better to visualize a portion of the graph instead of the whole graph. One way to do this in a straight-line drawing is to do a two step triangulation. The selected subgraph is first triangulated and then the rest of the graph is added and triangulated. This makes the selected subgraph to be drawn disregarding the rest of the graph. But this may not always be possible because of planarity constraints. In these cases another technique can be used to pop the subgraph from the whole graph. The subgraph is separately drawn and softer color tones are used in the rest of the graph.

4.4 Multi-arcs

As mentioned before FAgoo specially focuses in finite automata diagrams. These type of graphs often use multi-arcs in their representation. However general graph drawing algorithms do not support multi-graphs and simplify them in a early stage. This may have bad repercussions during the recovery the original graph. When the original graph is recovered multiple arcs may override other nodes or even arcs. A way to overcome this problem is to replace every multiple arc by a new node with arcs to the original arc source and target. The Fig. 5 illustrates this step. Then when recovering the original graph the created nodes are used as control points for the arcs splines. This way no multiple arcs will override other nodes or arcs. The graph planarity is also obviously not affected.

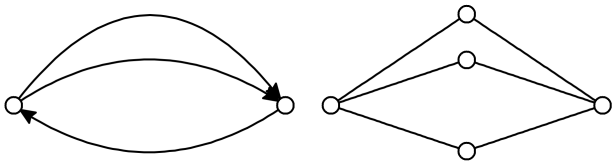


Fig. 5. Multi-arcs step illustration.

4.5 Force Directed Model

An interesting approach to the automatic graph drawing problem as been the simulation of forces. Increasingly force directed algorithms have been adopted by graph drawing libraries. The model used in FAgoo replaces the arcs with springs and the nodes with spheres. Between these spheres we introduce a repulsion force. The spheres are spread in a plane and the simulation stops when a equilibrium state is reached.

Reading directionality is achieved by fixing the initial states to the plane that is then lent to the right and gravity does the rest. This cause the other states to fall into to the right side of the initial states.

A graphical interface for this model is being developed. The idea of this interface is to allow the user to interact in real time with the ongoing simulation. The user can pause and resume the simulation to manually adjusting some components of the graph. For example, the user may want to fix the position of one or more nodes during a simulation or set specific strength values for some nodes and springs.

5 Conclusion

In this paper we presented GUItar as a tool for the visualization and editing of finite automata diagrams that combined with FAdo provides a potential graphical environment for automaton manipulation. We also presented the FFC mechanism that allows GUItar's expansion broaden the application scope to other type of graphs. We also presented FAgoo: a graph drawing library specialized in finite automata diagrams. FAgoo is integrated with GUItar using the FFC mechanism. Both GUItar and FAgoo are ongoing projects. FAgoo still needs improvements in some of the presented algorithms as well the development and implementation of many others.

References

1. FAdo Project: FAdo: tools for formal languages manipulation. `http://www.ncc.up.pt/FAdo` (Access date:12.06.2010)
2. aiSee Graph Layout Software: aiSee. `http://www.aisee.com/` (Access date:12.06.2010)

3. yWorks GmbH: yWorks. http://www.yworks.com/ (Access date:12.06.2010)
4. Chair of Algorithm Engineering, Juniorprofessorship of Algorithm Engineering, C.o.P.J., oreas GmbH: Open Graph Drawing Framework. http://www.ogdf.net/doku.php (Access date:12.06.2010)
5. Labs, A.R.: Graphviz - Graph Visualization Software. http://www.graphviz.org/ (Access date:12.06.2010)
6. Rodger, S.H., Finley, T.W.: JFLAP: An interactive formal languages and automata package. Jones & Bartlett Publishers (2006)
7. Smart, J., Roebling, R., Zeitlin, V., Dunn, R.: wxWidgets 2.6.3: A portable C++ and Python GUI toolkit. (2006)
8. WWW Consortium: XML specification WWW page. http://www.w3.org/TR/xml (Access date:12.06.2010)
9. Alves, J., Moreira, N., Reis, R.: XML description for automata manipulations. In Simões, A., Cruz, D., Ramalho, J.C., eds.: Actas XATA 2010, XML: aplicações e tecnologias associadas, ESEIG, Vila do Conde (2010) 77–88
10. GraphML Working Group: The GraphML file format. http://graphml.graphdrawing.org (Access date:12.06.2010)
11. Graph Visualization Software: The dot language. http://www.graphviz.org (Access date:12.06.2010)
12. Lombardy, S., Sakarovitch, J.: Vaucanson-G. http://igm.univ-mlv.fr/~lombardy (Access date:1.12.2009)
13. WWW Consortium: XSLT specification WWW page. http://www.w3.org/TR/xslt (Access date:12.06.2010)
14. Purchase, H.C., Cohen, R.F., James, M.I.: Validating graph drawing aesthetics. In: Graph Drawing. (1995) 435–446
15. Hopcroft, J.E., Tarjan, R.E.: Efficient planarity testing. J. ACM **21**(4) (1974) 549–568
16. Mehlhorn, K., Mutzel, P., Naher, S.: An implementation of the Hopcroft and Tarjan planarity test and embedding algorithm. Technical report, Research Report MPI-I-93-151, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 (1993)
17. Kant, G.: Algorithms for Drawing Planar Graphs. PhD thesis, Universiteit Utrecht, Faculteit Wiskunde en Informatica (1993)
18. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica **10**(1) (1990) 41–51
19. Read, R.C.: A new method for drawing a planar graph given the cyclic order of the edges at each vertex. Congressus Numerantium (56) (1987) 31–44
20. Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. SIAM J. Algebraic Discrete Methods (4(3)) (1983) 312–316
21. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness (1979)

Instant Global Illumination on the GPU using OptiX

Ricardo Marques and Luís Paulo Santos

Universidade do Minho, Braga, Portugal,
ricjmarques@gmail.com, psantos@di.uminho.pt

Abstract. OptiX, a programmable ray tracing engine, has been recently made available by NVidia, relieving rendering researchers from the idiosyncrasies of efficient ray tracing programming and allowing them to concentrate on higher level algorithms, such as interactive global illumination. This paper evaluates the performance of the Instant Global Illumination algorithm on OptiX as well as the impact of three different optimization techniques: imperfect visibility, downsampling and interleaved sampling. Results show that interactive frame rates are indeed achievable, although the combination of all optimization techniques leads to the appearance of artifacts that compromise image quality. Suggestions are presented on possible ways to overcome these limitations.

Keywords: instant global illumination, ray tracing, graphics processors

1 Introduction

Interactive ray tracing became possible along the last decade on both CPU and GPU based platforms. However, this has been achieved through extensive optimization of code and data structures, thus developing such a ray tracer is a complex and time consuming task. In September 2009 Nvidia launched a programmable ray tracing engine for their GPUs, OptiX [1], which allows researchers to concentrate on higher level algorithms while still being able to trace rays efficiently.

The goal of this paper is to assess the performance of an interactive global illumination (GI) algorithm on OptiX. This algorithm, referred to as Instant Global Illumination [2], computes indirect diffuse interreflections by generating a particle based approximation of this illumination component, resulting in a three-dimensional distribution of secondary virtual point light (VPL) sources. The algorithm in its original form is barely interactive due to the high number of VPLs. Optimizations have been proposed under the form of imperfect visibility [3], downsampling the indirect diffuse evaluation rate [4] and interleaving VPL sampling patterns [5]. This paper assesses the performance achieved with these three optimizations on a last generation NV 480 GTX GPU using OptiX and proposes a few hypothesis for further performance gains.

The next section presents related work and details on the optimization techniques. Section 3 briefly introduces OptiX, while the used algorithm is detailed

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 329–340

in section 4. Results are analyzed in section 5. The paper concludes with some suggestions for future work.

2 Related Work

2.1 Interactive Ray Tracing and Global Illumination

Interactive Whitted-style [6] ray tracing (iRT) became possible along the first decade of the XXIst century, for both static and dynamic scenes, through clever exploitation of advances in available computing power and careful optimization of both code and used data structures [7, 8]. The performance of such ray tracers arises mainly from fine tuning data structures such that the memory hierarchy performs to its maximum and by exploiting ray coherence: rays are grouped into packets or frusta and SIMD instructions are used to trace them through the scene. Ray coherence exploitation has been shown to be effective for primary and shadow rays. Whether it can be used to speedup tracing of secondary rays is still unclear, since these often do not share the same origin and exhibit less directional coherence [9].

Ray tracing is an embarrassingly parallel algorithm which naturally leads to the exploitation of parallel systems. Most the above cited approaches exploit parallelism at several levels: SIMD, multicore and clusters of distributed memory machines. With the computation power and core count of GPUs increasing at a higher rate than those of CPUs, ray tracing solutions that harness the GPUs processing capabilities begun to emerge [10, 11]. However, even with the appearance of flexible, C-like, GPU programming languages such as CUDA [12], efficient programming of these devices is still not straightforward due to their SIMT (Single Instruction Multiple Threads) computing model. NVIDIA has recently made available a GPU ray tracing engine, OptiX [1], which relieves researchers from the idiosyncrasies of efficient ray tracing programming and allows them to concentrate on higher level algorithms, such as global illumination.

While Whitted style ray tracing requires tracing a reasonable number of mostly coherent rays (specular rays may exhibit less coherence, specially over curved surfaces), GI entails simulating a huge number of incoherent light paths, thus making it hard to maintain interactive frame rates. GI light transport phenomena include diffuse interreflections, caustics and participating media. Indirect diffuse interreflections, the focus of this paper, are typically simulated using Monte Carlo path tracing [13], photon mapping [14], irradiance caching [15] or instant radiosity [16]. The later is frequently used for interactive rendering [2, 17]: it generates a particle approximation of the indirect diffuse radiant scene by performing quasi-random walks on a quasi-Monte Carlo integration framework. Photons are traced from the light sources into the scene and Virtual Point Light sources (VPLs) are placed at the intersections of the quasi-random photon paths with diffuse geometry. The image is then rendered by sampling the VPLs as point light sources. This algorithm exhibits ray coherence similar to direct lighting and is thus expected to perform similarly on vectorial processors such as the GPUs.

2.2 Accelerating Indirect Diffuse

The quality of the indirect diffuse estimate is dependent on the number of VPLs and the quality of their distribution throughout the scene. Unfortunately, rendering time is linearly proportional to the number of VPLs, which requires clever strategies to speedup indirect diffuse calculations.These strategies are based on two key observations: indirect diffuse lighting is mostly a low frequency signal that varies smoothly over the scene and accurate visibility is not required for indirect illumination.

Accurate visibility is traditionally used in light transport at the cost of tracing rays against the detailed scene description. However, indirect diffuse illumination varies smoothly across the scene, thus accurate visibility is perceptually unnecessary in this case, since visibility errors are masked by the low frequency nature of the signal [18]. This insight has been exploited by either performing accurate visibility queries only on the neighborhood of the shading point [19] or by testing visibility against a crude representation of the scene [3].

Instant radiosity, on its original form, requires evaluating VPLs visibility at each shading point. The number of shading points is thus linearly correlated with the number of pixels. By taking advantage of the low frequency nature of indirect diffuse reflections, the estimate can be computed at a lower image resolution and then upsampled to the target resolution [4]. Despite being mostly smooth, high frequencies are still present on the indirect diffuse signal, mostly due to geometric discontinuities. Upsampling must thus be done using some discontinuity preserving filter, such as the joint bilateral filter, which uses geometric information obtained at full resolution when computing direct illumination.

Sampling the whole set of VPLs for each pixel is expensive and might compromise performance. Interleaved sampling [5] has thus been proposed to accelerate instant radiosity resulting on the so-called Instant Global Illumination algorithm [2]. The set of VPLs is divided onto $m * n$ subsets and for each pixel within a $m * n$ tile of pixels only one of the subsets is sampled, spawning a much lower number of VPL shadow rays. Results are then integrated over each tile by using the discontinuity buffer.

3 OptiX

NVIDIA's OptiX engine is a programmable ray tracing pipeline for NVIDIA GPUs using the CUDA-C programming language [1, 20]. OptiX abstracts the execution to single rays, simplifying the application programmer's role, while internally exploiting the GPU architectural characteristics through deferred shading and built-in scheduling and load balancing. OptiX is tightly coupled with graphics APIs to allow combinations of raster and ray tracing approaches. The ray tracing pipeline is programmable through programmer supplied programs (CUDA kernels) that handle the various ray tracing events, such as intersections for procedurally accurate surface types, cameras for new composition potential, shading and scene graph traversal. OptiX includes support for parallelism across

multiple GPUs and building and traversal of acceleration structures (BVH and KD trees).

OptiX runs on most CUDA enabled GPUs although it is only fully functional and supported on the latest architectures (GT200 and GF100). Currently, there are two main drawbacks associated with OptiX: acceleration structure traversal is not programmable, which precludes their utilization on tasks other than ray traversal, and the CUDA context upon which OptiX runs is not visible to the programmer, prohibiting explicit access to shared and constant memory, which prevents the utilization of most common CUDA optimization techniques.

The availability of a programmable, high performance, ray tracing engine relieves application developers from the idiosyncrasies of efficient ray tracing programming, allowing them to concentrate on higher level algorithms such as global illumination. However, applications using OptiX must still be carefully designed and optimised if high performance is to be achieved.

4 The Algorithm

The algorithm proposed on this paper simulates direct illumination, specular reflections and indirect diffuse interreflections using the Instant Global Illumination approach [2]. Figure 1 illustrates the fundamental stages of the proposed pipeline; the upper arrow depicts the temporal order of the different stages, while the arrows connecting the boxes illustrate data dependencies.

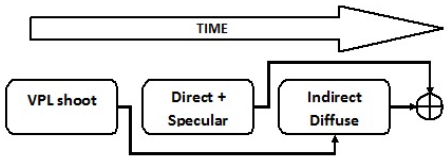


Fig. 1. Rendering pipeline for the canonical version of the algorithm

Particles are shot from the light sources following a quasi-random Halton sequence. VPLs are then created at each intersection of the particles path with geometric primitives whose material has a diffuse component. The number of bounces along each path is a user supplied parameter. The OptiX context launched to shoot the VPLs consists on as many threads as the number of paths, each thread processing a whole path. The quasi-random numbers used to build the particle paths are generated on the CPU and passed to the GPU as OptiX buffers.

Direct plus specular illumination entails shooting one primary ray per pixel, spawning a Whitted-style tree of rays. Only point light sources are supported. Indirect diffuse is only separated from direct plus specular at the primary ray hit point. Hit points further down the rays' tree, resulting from tracing specular rays,

have their indirect diffuse component calculated within the direct plus specular stage of the pipeline. This approach was selected to exploit OptiX recursion capabilities. Explicit separation of components could be implemented by storing the hitpoints on a buffer, but this would increase access to global memory thus increasing rendering time (shared memory can not be explicitly accessed from within the OptiX context).

Indirect diffuse radiance, $L_{indirect}(x)$, is evaluated by shooting, at each shading point x, one ray towards each VPL to assess its visibility:

$$L_{indirect}(x) = \sum_{k=1}^{N} \rho(x) L_{e,k} V(x, y_k) G(x, y_k) \tag{1}$$

where N is the number of VPLs, $V(.,.)$ is the visibility function between two points, $L_{e,k}$ is the emitted radiance for the k-th VPL, y_k is the position of the k-th VPL, ρ is the diffuse reflectance coefficient and G is the bounded geometry term, defined as

$$G(x, y_k) = \frac{cos\theta_x cos\theta_{y_k}}{\|x - y_k\|^2} f(0.8min_d, 1.2min_d, \|x - y_k\|)$$

where θ_x and θ_{y_k} are the angles between the normal at x, respectively y_k, and the direction $x \rightarrow y_k$, min_d is the bounding distance (to avoid singularities in G) and $f(a, b, d)$ is a smoothing function returning 0 if $d < a$, 1 if $d > b$ and a quadratic value in the interval [0..1] if $a <= d <= b$.

Equation 1 is quite expensive to compute since visibility has to be evaluated for all the VPLs and shading points. This motivates the acceleration strategies proposed on the next subsections.

4.1 Imperfect Visibility

It has been shown that due to the low frequency nature of the indirect diffuse illumination accurate visibility is not perceptually important [18]. Within our instant radiosity inspired approach this means that assessing the term $V(.,.)$ on equation 1 can be relaxed in an attempt to reduce rendering times. To validate this hypothesis VPL shadow rays are tested against the triangles bounding boxes rather than testing them for intersection against the triangles themselves.

The application supplied intersection program (or kernel) used by OptiX is associated with the geometric primitive type and does not depend on the ray type. In practice, this means that it is not possible to have OptiX calling a given intersection program for all ray types and then another program, that would test the ray against the triangle bounding box rather than the triangle itself, for VPL shadow ray types. In order to use a different intersection algorithm there are two alternatives: either a conditional statement is included on the general intersection program or the scene graph is duplicated within the OptiX context and the new intersection program is associated with the second scene graph. The former has the disadvantage of implying evaluating the conditional statement for all

intersection tests and, worst, can lead to execution divergence due to the GPU's SIMT computation paradigm. The latter has the disadvantage of consuming more memory (although only the bounding box coordinates are stored, not the respective triangle vertices) and requires building a second acceleration structure - on dynamic scenes this might compromise interactivity. Since dynamic scenes are currently not supported, the second approach was selected and a second scene graph is built with the triangles' bounding boxes rather than the triangles themselves.

4.2 Downsampling

Typically, indirect diffuse illumination is computed for all shading points. Image resolution, however, continues to grow every year with advances in available computing power, storage space and display capabilities. Since rendering complexity is linear in time with the number of pixels, computing indirect illumination at such high resolution prevents interactivity. The fact that the indirect diffuse component is mostly a low frequency signal can be exploited by rendering it at a lower resolution and then upsampling to the target final resolution [4].

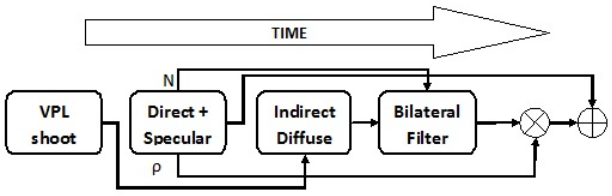


Fig. 2. Rendering pipeline for indirect upsampling - the direct stage contributes with full resolution normal and ρ maps for filtering and composition

The indirect diffuse signal, however, still has high frequencies, mostly due to geometric discontinuities. Upsampling can not be performed by convolving the signal with some low-pass kernel, since sharp edges would be unacceptably blurred. Since a high resolution pass is still required to compute direct plus specular illumination, this can be used to gather geometric information about each pixel in the target image (see figure 2). This information, the normal at the intersection point, can be used during upsampling to properly weight the contribution of each neighbor to the final value. The reasoning is that pixels in the neighborhood which have similar orientations to the center pixel will contribute more to its final result. We use the joint bilateral filter to perform this task: a spatial filter is applied to the low resolution image I and a range filter is applied to the full resolution image $\tilde{I}$. Let $\tilde{p}$ and $\tilde{q}$ denote the coordinates of two pixels in $\tilde{I}$, and p and q denote the corresponding coordinates in the low

resolution solution I. The upsampled solution $\tilde{S}$ is

$$\tilde{S}_{\tilde{p}} = \frac{1}{k_{\tilde{p}}} \sum_{q \in \Omega} I_q f(|p - q|)(\boldsymbol{N}_{\tilde{p}} \cdot \boldsymbol{N}_{\tilde{q}}) \tag{2}$$

where f is the spatial Gaussian kernel centered over p, Ω is the spatial support of f and $k_{\tilde{p}}$ is a normalizing constant. The range filter is the cosine of the angle of the normals at $\tilde{p}$ and $\tilde{q}$.

The indirect diffuse stage computes incident indirect radiance, rather than reflected, such that the upsampling filter does not blur details due to local material properties (e.g., mapped textures). Multiplication by the diffuse reflection coefficient, ρ, is done just before composition.

The indirect diffuse stage computes indirect illumination only for the shading points determined by the primary rays. Thus, upsampling is only applied to these points. Shading points further down the tree of rays have their indirect illumination calculated by the direct stage, which operates at full resolution.

4.3 Interleaved Sampling

In order to further reduce the number of VPLs visibility queries interleaved sampling is applied [2, 5]. The VPLs are divided into 9 subsets and within each pixel of a $3 * 3$ tile a different subset is used to compute indirect illumination. The contributions of the different VPL subsets are then integrated using the discontinuity buffer. The difference of depths and the dot product of the normals of a pixel are compared to those of each of its 8 neighbors. If both these values are below some given thresholds, then geometry is considered locally continuous and incident irradiance from that neighbor is added to the center pixel. The final indirect incident value is evaluated by dividing by the number of neighbors that contributed (including the center pixel itself).

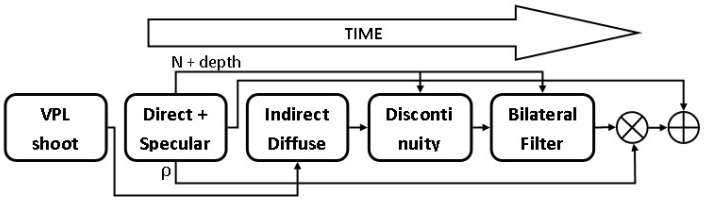


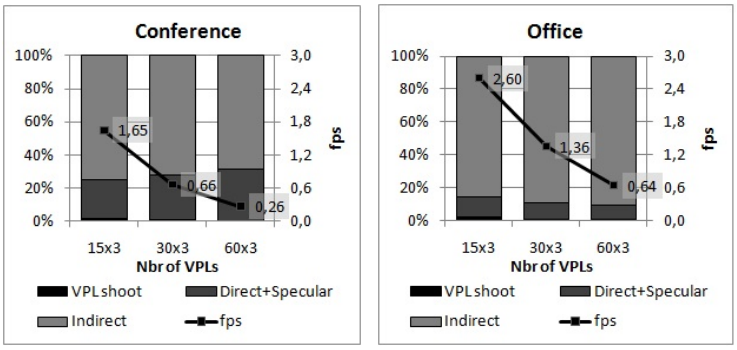
Fig. 3. Rendering pipeline for upsampling and interleaving - the direct stage contributes with full resolution normal, depth and ρ maps for filtering and composition

5 Results

5.1 Experimental Setup

All experiments and measurements were performed using OptiX 2 Beta 5 and Visual Studio 2008 on a dual core Intel Xeon 3.20 Ghz machine with 2 GB of memory and the new Nvidia 480 GTX GPU with 4 GB of RAM. Two scenes were used: the Conference room (190K triangles, 4 point light sources) and Office (21K triangles, 2 point light sources). Images were rendered at a resolution of 800x600 pixels, no anti-aliasing, using OptiX built-in BVH as the acceleration structure. Time measurements were performed with 30, 90 and 180 VPLs. Where appropriate the downsampling window was 4x4 and interleaving was 3x3.

5.2 Results Analysis



(a) Conference (b) Office

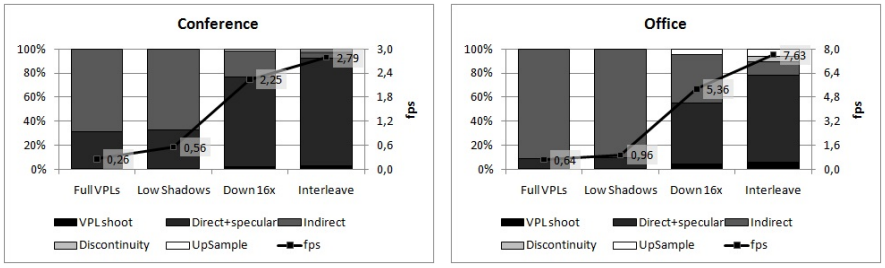
Fig. 4. fps and time percentage spent on each illumination component

Figure 4 shows evaluation of the indirect diffuse component dominates rendering time and that this aggravates with the number of VPLs. Although not obvious for the conference scene (figure 4(a)), because it contains many specular objects, the dependance on the number of VPLs for the office scene is quite obvious. Thus, according to Amdahl's law this component is the one that is worth optimizing.

Figure 5 shows the frame rate and relative execution time for each of the rendering and filtering kernels (see also figures 6 and 7).

Imperfect visibility achieves a speedup between 1.5 and 2.0 without any perceptually significant impact on the rendered image. This technique accelerates all indirect diffuse calculations, including those triggered by secondary rays - it has thus a most significant impact on the conference scene.

Downsampling 16 times provides a speedup of approximately 5 times without significantly impacting on the quality of the rendered images. Artifacts due to



(a) Conference (b) Office

Fig. 5. fps and time percentage spent on each illumination component for the 4 different approaches and 180 VPLs

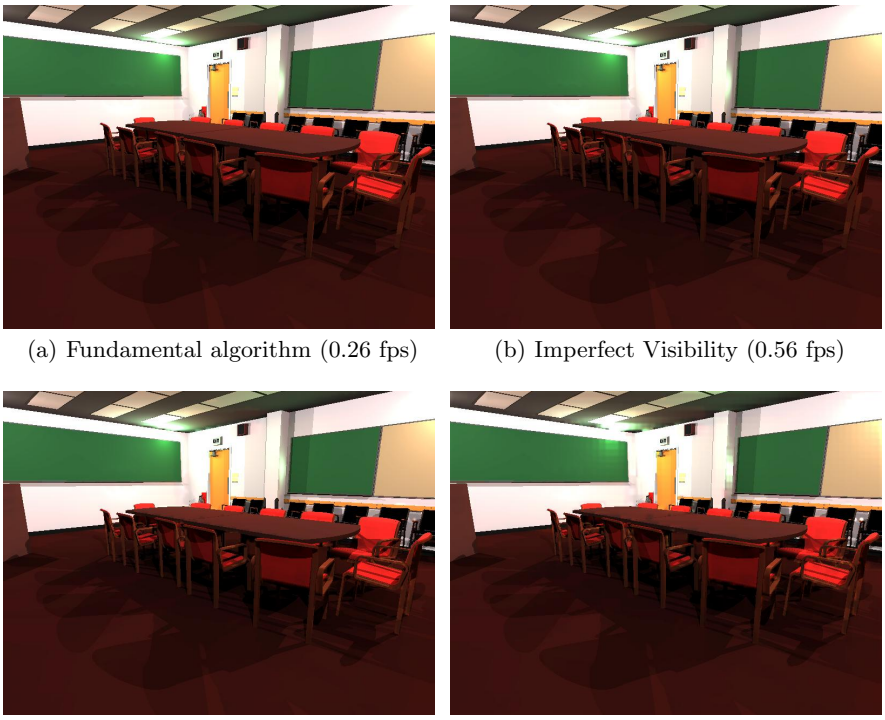
incorrect geometric continuity assumptions are however visible along edges. See for example the junction of the two halves of the table top in figure 6(c). The indirect diffuse rendering time is so drastically reduced that it is no longer the bottleneck: direct plus specular now takes more than 50% of the total rendering time. Note, however, that indirect calculations triggered by secondary rays are included in the direct stage and are not optimized by either downsampling or interleaving. In scenes with reflective materials, such as the conference room, this contributes to increase this stage relative weight on the total rendering time.

Interleaving VPLs sampling over a 3x3 window further accelerates indirect diffuse evaluation. However, artifacts are now perceivable, most noticeably when the indirect component has a strong contribution, such as on the Conference ceiling and under the desk in the Office scene (figures 6(d) and 7(d)). These artifacts are due to the regular interleaved sampling pattern over the image plane and are further enhanced by the upsampling step. Minimization of such artifacts might be possible by reducing the interleaving window size (e.g., 2x2) or by using an irregular interleaving pattern [21]. The gains obtained with interleaving are not enough to compensate for the added artifacts; furthermore, the direct plus specular component together with secondary indirect diffuse calculations now dominate rendering times, thus optimizing these is probably more important than applying interleaved sampling.

6 Conclusions

This paper discusses an implementation of Instant Global Illumination over OptiX and then evaluates three acceleration techniques: imperfect visibility, downsampling and interleaved sampling. Results show that these techniques combined allow for interactive rendering of relatively complex scenes (e.g., conference room, 190 K triangles, 180 VPLs), achieving up to 2.8 fps.

Imperfect visibility is straightforward to put in practice, but it is downsampling that results in the most impressive performance gains; artifacts are, however, slightly perceived throughout the image. Reducing the downsampling rate,



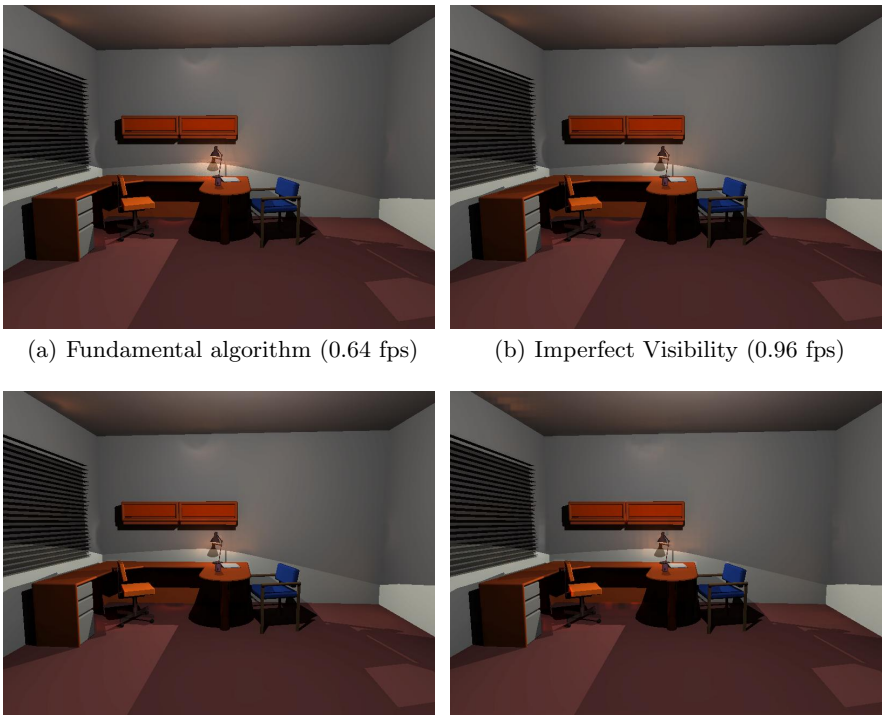
(a) Fundamental algorithm (0.26 fps) (b) Imperfect Visibility (0.56 fps)

(c) 4 x 4 downsampling (2.25 fps) (d) 3 x 3 interleaved (2.79 fps)

Fig. 6. Images for the conference scene - 180 VPLs

e.g. 3x3, will reduce such artifacts at some performance cost. Interleaved sampling further contributes to reduce rendering times, but, in combination with upsampling, artifacts become too obvious due to the regular interleaved sampling pattern. These two last techniques only operate on indirect diffuse calculations triggered by primary rays; those associated with specular secondary rays are neither downsampled nor interleaved. Improving such secondary irradiance calculations would significantly enhance performance in scenes with reflective materials and will be addressed as future work. We propose to adopt the instant caching approach [22], which, similarly to the irradiance cache [15], interpolates over scene space thus accelerating all indirect diffuse calculations.

Future work will include using irregular interleaved patterns and combining the discontinuity buffer and the bilateral filter in a single pass. Also combining rasterization with ray tracing, by resoting to shadow maps rather than shadow rays, might result in significant performance gains. Since OptiX and OpenGL interoperability is assured by NVidia, this should be straightforward to implement and evaluate. Finally, OptiX does not allow explicit access to CUDA shared memory, which results on implementation penalties, particularly on operations such as filtering. However, data can be passed between CUDA contexts and Op-



(a) Fundamental algorithm (0.64 fps) (b) Imperfect Visibility (0.96 fps)

(c) 4 x 4 downsampling (5.36 fps) (d) 3 x 3 interleaved (7.63 fps)

Fig. 7. Images for the office scene - 180 VPLs

tiX contexts through GL buffers; we intend to use this feature to optimize the discontinuity and bilateral filters, which might allow for the utilization of more sophisticated discontinuity detection techniques.

Acknowledgements This work was partially funded by PT-FCT grant PTDC/ EIA/ 65965/ 2006 (IGIDE project: Interactive Global Illumination within Dynamic Environments)

References

1. Steven Parker. Efficient ray tracing on nvidia gpus. SIGGRAPH ASIA 2009 Presentation, December 2009.
2. Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. Interactive global illumination using fast ray tracing. In *EGRW '02: Proceedings of the 13th Eurographics Workshop on Rendering*, 2002.
3. T. Ritschel, T. Grosch, M. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 27(5), 2008.

4. Johannes Kopf, Michael Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Transactions on Graphics*, 26(3), 2007.
5. Alexander Keller and Wolfgang Heidrich. Interleaved sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 269–276, London, UK, 2001. Springer-Verlag.
6. Turner Whitted. An improved illumination model for shaded display. *Communicantions of the ACM*, 23(6):343–349, 1980.
7. Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent raytracing. In *Computer Graphics Forum/Proceedings of EUROGRAPHICS*, volume 20, pages 153–164, 2001.
8. I. Wald, W. R. Mark, J. Günther, S. Boulos, Ize T, Hunt W, S. Parker, and P. Shirley. State of the art in ray tracing animated scenes. In *STAR Proceedings of Eurographics 2007*, pages 89–116, September 2007.
9. S. Boulos, D. Edwards, J. Lacewell, J. Kniss, J. Kautz, I. Wald, and P. Shirley. Packet-based whitted and distribution ray tracing. In *Graphics Interface*, 2007.
10. Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on gpus. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, pages 145–149, New York, NY, USA, 2009. ACM.
11. Min Shih, Yung-Feng Chiu, Ying-Chieh Chen, and Chun-Fa Chang. *Algorithms and Architectures for Parallel Processing*, volume 5574 of *LN in Computer Science*, chapter Real Time Ray Tracing with CUDA, pages 327–337. 2009.
12. David Kirk and Wen mei Hwu. *Programming Massively Parallel Processors: a Hands-on Approach*. Korgan Kaufmann, 2010.
13. James T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM.
14. Henrik Wann Jensen. Global illumination using photon maps. In X. Pueyo and P. Schröder, editors, *Rendering Techniques*, pages 21–30. Springer-Verlag, 1996.
15. G. Ward, F. Rubinstein, and R. Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics*, 22(3), 1988.
16. Alexander Keller. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.
17. Rui. Wang, R. Wang, K. Zhou, M. Pan, and H. Bao. An efficient gpu-based approach for interactive global illumination. *ACM Trans. Graph.*, 28(3):1–8, 2009.
18. I. Yu, A. Cox, M. Kim, T. Ritschel, T. Grosch, C. Dachsbacher, and J. Kautz. Perceptual influence of approximate visibility in indirect illumination. In *ACM Transactions on Applied Perception*, volume 6, 2009.
19. Okan Arikan, David Forsyth, and James O'Brien. Fast and detailed approximate global illumination with irradiance decomposition. *ACM Transactions on Graphics (ACM SIGGRAPH 2005)*, pages 1108–1114, 2005.
20. Holger Ludvigsen and Anne Cathrine Elster. Real-time ray tracing using nvidia optix. In *Eurographics 2010 short papers*, 2010.
21. Solomon Boulos, Dave Edwards, Dylan Lacewell, Joe Kniss, Jan Kautz, Peter Shirley, and Ingo Wald. Interactive distribution ray tracing. Technical Report UUSCI-2006-022, SVCI Institute, University of Utah, 2006.
22. K. Debattista, P. Dubla, F. Banterle, L.P. Santos, and A. Chalmers. Instant caching for interactive global illumination. *Computer Graphics Forum*, 28(8):2216–2228, 2009.

Projecções Interactivas na Sala de Aulas

Vasco M. A. Santos[1] and Frutuoso G. M. Silva[2]

Instituto de Telecomunicações, Universidade da Beira Interior, Covilhã,
[1]m2059@ubi.pt,[2]fsilva@di.ubi.pt,
WWW home page: http://regain.it.ubi.pt/

Resumo Hoje em dia, o uso das Tecnologias de informação e Comunicação (TIC) tem tido, cada vez mais, um papel chave nas escolas e nos métodos de ensino. O uso de quadros interactivos está em profunda expansão. Os estudantes passam a ter as notas em formato digital em vez do caderno, permitindo uma maior partilha de recursos e ideias entre estudantes e professores. No entanto, o facto das soluções disponibilizadas pelos grandes fabricantes terem um preço ainda alto, limitando o acesso a esses recursos a um grande número de instituições de ensino, levou a que se procurassem soluções mais baratas. Uma dessas soluções é o Wiimote Whiteboard apresentdo por Johnny Lee que permite que qualquer pessoa tenha um quadro interactivo de baixo custo, usando apenas um Wiimote e um emissor de infravermelhos. Com base nesta tecnologia, criámos três aplicações de suporte às aulas para uso conjunto com o Wiimote Whiteboard. Estas aplicações serão descritas, mostrando as principais diferenças para com as soluções existentes no mercado. Além disso, apresenta-se um novo método para controlar o computador à distância, o qual permite uma maior liberdade ao apresentador.

Abstract Nowadays, the use of Information and Communication Technology (ICT) has had, increasingly, a key role in schools and the teaching methods. The use of interactive whiteboards is in deep expansion. Students come to have the notes in digital format instead of the notebook, allowing greater sharing of resources and ideas between students and teachers. However, the fact that the solutions offered by major manufacturers have a high price, limiting access to these resources to a large number of educational institutions, meant that there was a demand for cheaper solutions. One such solution is the Wiimote Whiteboard presented by Johnny Lee that allows anyone to have a low cost interactive whiteboard using only the Wiimote and an infrared emitter. Based on this technology, we have created three applications to support classes for use in combination with the Wiimote Whiteboard. These applications will be described, showing the main differences for identical solutions existing in the market. Besides, we present a new way to control the computer at distance that gives more freedom to presenter.

1 Introdução

Os métodos de ensino na sala de aula estão em constante evolução. Os professores recorrem cada vez mais a apresentações (ex: em PowerPoint) e a quadros

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 341–352

interactivos, em vez dos antiquados acetatos e quadros de giz, disponibilizando apontamentos aos estudantes em formato digital através da Internet. Aplicações que permitam ao professor fazer anotações sobre o que é apresentado e permitam depois disponibilizar essas anotações para impressão, fazem com que os estudantes se concentrem mais na apresentação e se preocupem menos em tirar anotações.

Em 2001, os países da União Europeia (UE) definiram como objectivo o aumento da qualidade e eficácia da educação e aprendizagem na UE, do qual resultaram duas questões-chave [1]:
(1) fornecer equipamento adequado e software educativo para optimizar o uso das TIC e os processos de e-Learning na educação e ensino e;
(2) encorajar o melhor uso das técnicas de ensino e aprendizagem inovadoras baseadas nas TIC.

Mais tarde, um estudo realizado em Inglaterra [2] avaliou o impacto das TIC na educação e, mais especificamente, o impacto dos quadros interactivos nos estudantes e professores, e as barreiras que surgiram. Ao nível dos estudantes e da sua aprendizagem, o estudo mostrou que:

- as TIC têm um impacto positivo no desempenho educacional nas escolas primárias, em especial no Inglês, mas menos nas Ciências, com a excepção da Matemática;
- o uso das TIC aumenta o desempenho dos estudantes em Inglês (como língua materna), Ciências, Desenho e Tecnologia entre as idades dos 7 e 16 anos, especialmente nas escolas primárias;
- nos países da OCDE (Organização para a Cooperação e Desenvolvimento Económico), existe uma associação positiva entre a quantidade de tempo de utilização das TIC e o desempenho dos estudantes nos testes de Matemática do PISA (Program for International Student Assessment);
- escolas com bons recursos de TIC alcançam melhores resultados que as escolas mal equipadas;
- a introdução de quadros interactivos nas salas de aula aumentou o desempenho dos estudantes nos exames nacionais de Inglês, Matemática e Ciências mais do que o desempenho dos alunos de escolas sem quadros interactivos.

Além disto, foram também observados benefícios nos estudantes em relação à motivação e competências, aprendizagem independente e trabalho de equipa.

Ao nível dos professores e ensino, o estudo mostrou um conjunto considerável de provas do impacto das TIC, nomeadamente:

- um maior entusiasmo ao ensinar os alunos;
- aumento da eficiência e colaboração dos professores;
- os quadros interactivos fazem a diferença em aspectos de interacção na sala de aulas entre alunos e professores;
- aumento das competências dos professores no uso das TIC.

O objectivo do nosso trabalho foi implementar um quadro interactivo de baixo custo baseado na proposta apresentada por Johnny Lee e desenvolver novas maneiras de interacção humana com o computador à distância. Por isso,

desenvolveram-se aplicações de suporte às aulas que permitem ao professor/apresentador utilizar o quadro interactivo como um quadro normal.

Este artigo está estruturado da seguinte forma: na secção 2 é feita uma breve descrição das tecnologias disponíveis no mercado em relação aos quadros interactivos; na secção 3 é descrita a nova forma de interacção à distância entre o utilizador e o computador, bem com as aplicações desenvolvidas para tornar as projecções interactivas; na secção 4 serão apresentadas as conclusões e trabalho futuro.

2 Quadros Interactivos

Um quadro interactivo é um dispositivo ligado a um computador e a um vídeo projector [3]. O projector projecta a imagem do ecrã do computador na área onde o utilizador pode interagir usando o dedo, uma caneta ou outro dispositivo. Existem 3 tipos de quadros interactivos:

- Os quadros interactivos de projecção frontal têm o vídeo projector em frente ao quadro. Uma desvantagem destes quadros é a sombra que o utilizador pode provocar durante a apresentação devido ao facto de se colocar entre o projector e o quadro. Entretanto, colocar o projector numa posição alta pode minimizar esta desvantagem. Outra desvantagem é o facto do utilizador poder ficar encadeado com a luz do projector enquanto fala para a assistência;
- Os quadros interactivos de projecção traseira têm o vídeo projector localizado atrás do quadro de modo a remover sombras. Deste modo, o utilizador não está sujeito a ser encadeado pela luz do projector enquanto fala para a assistência. No entanto, este tipo de quadros interactivos são muito mais caros e ocupam mais espaço, visto que não podem ser montados numa parede. Entretanto, é possível embutir estes quadros numa parede de modo a não ocupar tanto espaço;
- O painéis planos [4] são quadros interactivos em que a área de interacção é um monitor LCD ou plasma.

Recentemente tem havido um grande interesse na educação para integrar os quadros interactivos nos métodos de ensino. Isto levou a desenvolvimentos positivos em relação à sua utilização, nomeadamente porque [5]:

- Facilitam a colaboração com colegas e parceiros;
- Recorre-se a desenhos para que a turma possa visualizar em conjunto, pois a informação visual é partilhada e entendida mais facilmente;
- Aumentam a motivação dos alunos, pois estes gostam de interagir fisicamente com o quadro, manipulando texto e imagens, fornecendo mais oportunidades para interacção e discussão;

São ainda notadas vantagens psicológicas como o aumento do planeamento e preparação, na marcação e avaliação, em guardar e editar lições, no estilo de ensino, na sensibilização de estilos de ensino, no planear para o desenvolvimento

cognitivo, na clara representação visual de conceitos e nas actividades que encorajam uma abordagem de pensamento activa. Podem ser encontrados vários aspectos positivos associados a:

- *Compromisso*: há um aumento na motivação, credibilidade, validade e concentração da turma;
- *Aspectos socioculturais*: contribuem para uma melhor interacção social e um melhor trabalho de equipa;
- *Tecnologia*: o recurso a tecnologias como *drag-and-drop*, *esconder-e-revelar*, *faça-a-correspondência* e a utilização de movimento são boas maneiras de interacção entre os alunos e o quadro interactivo.

No entanto, devido às grandes limitações dos quadros interactivos, como o alto preço (ver Tabela 1) e a mobilidade, existe uma necessidade urgente de novas tecnologias para encontrar soluções que tenham um nível de desempenho similar, mas com um custo muito mais baixo. Uma solução existente é o sistema eBeam [6]. O dispositivo receptor de sistema eBeam é um dispositivo compacto, portátil e fácil de utilizar que torna qualquer superfície lisa num quadro interactivo. O sistema interactivo eBeam pesa menos de 200g, é instalado em minutos pois é amovível. Este sistema elimina a desvantagem da portabilidade, mas não o alto preço, pois custa 665€ sem o projector. Uma solução idêntica ao sistema eBeam que apareceu recentemente no mercado é o sistema mimio Interactive [7] que custa cerca de 595€ sem o projector. Apesar destes dispositivos já terem um preço em conta, comparado com um quadro interactivo tradicional ainda têm um preço alto quando se pensa em equipar várias salas de aula (i.e., nº salas × 600€).

Modelo	Reconhecimento de escrita	Projecção	Projector incluído	Preço
eBeam Integral 65	Não	Frontal	Não	790€
InterWrite 1071	Sim	Frontal	Não	1142€
Activboard 95 studio	Sim	Frontal	Não	1890€
SMARTBoard ESP680-N	Sim	Frontal	Sim	3390€
SMARTBoard 2000i	Sim	Traseira	Sim	7090€

Tabela 1. Características de cinco quadros interactivos (retirado de [5]).

Para fazer face às limitações dos quadros interactivos referidas anteriormente (i.e., preço alto e mobilidade) surgiu recentemente uma solução baseada numa caneta com um emissor de infravermelhos (IV) e uma câmara de IV. Neste caso a caneta de IV que feita com os componentes mais básicos fica com um custo a rondar os 5€, e para a câmara podemos usar o comando Wiimote, que custa cerca de 40€. Este sistema de nome Wiimote Whiteboard foi apresentado por Johnny Lee e teve uma ampla divulgação pela Internet [8].

O Wiimote é o comando da consola Nintendo Wii e pode ser ligado a qualquer computador através de Bluetooth, tal como um telemóvel. Entre várias características, o comando tem uma câmara de IV incorporada, capaz de detectar até quatro pontos emissores de IV, com uma resolução de 1024x768 pixeis, uma taxa de actualização de 100Hz e um ângulo de visão de 45º na horizontal [5,8,9]. Entretanto, a câmara sofre do mesmo problema dos projectores, isto é, se o apresentador estiver entre a câmara e o emissor de IV, poderá detectar a posição do emissor incorrectamente, ou mesmo não a detectar. Se a projecção for frontal, e o projector for colocado numa posição alta, colocar o Wiimote junto do projector é uma boa solução (ver Figura 1). O sistema desenvolvido por Johnny

Figura 1. Posicionamento do Wiimote (retirado de [10]).

Lee permite ao utilizador calibrar a área de projecção sempre que o entender, usando quatro pontos de referência. Este fornece ainda alguma informação sobre o comando tal como, o estado da bateria, o total de fontes IV detectadas e a utilização de rastreamento. Também permite ao utilizador configurar a suavidade da detecção do emissor de IV, bem como activar ou desactivar o controlo do cursor. No entanto, este sistema tem uma grande limitação: só permite emular o evento de clique do botão esquerdo do rato. Assim um dos objectivos do nosso trabalho foi também tentar eliminar esta limitação. Uma aplicação que interage com o Wiimote Whiteboard é o Smoothboard [11]. O Smoothboard permite ao utilizador usar o quadro tal como se fosse um quadro interactivo, tirando notas directamente sobre o que é apresentado. No entanto, se o utilizador quiser

guardar as notas, o Smoothboard apenas permite ao utilizador guardá-las numa imagem JPEG através da captura de uma imagem do ecrã do computador.

3 O nosso sistema de projecções interactivas

Com base na tecnologia apresentada por Johnny Lee, desenvolvemos o nosso próprio sistema de modo a termos um quadro interactivo de baixo custo. Deste sistema fazem parte uma aplicação de suporte ao quadro interactivo, que permite ao utilizador fazer apontamentos sobre o que é apresentado e uma aplicação que permite editar, à posteriori, as notas capturadas durante a apresentação. Além disto, foi ainda desenvolvida uma aplicação visual de suporte às aulas de Introdução à Programação. Estas aplicações foram desenvolvidas em Windows Presentation Foundation [14].

Foi também desenvolvida uma aplicação em C# que permite ao utilizador controlar o computador à distância sem necessidade de usar o teclado e o rato. Isto foi alcançado usando um segundo Wiimote, no qual foram mapeadas algumas das funções chave do rato e teclado nos botões do Wiimote, como se ilustra na Figura 2. Sendo assim, o utilizador pode controlar a apresentação à distância, logo torna o quadro mais interactivo.

Figura 2. Teclas no Wiimote.

3.1 Aplicação iiNote

Para suportar o uso do quadro interactivo, decidiu-se implementar uma aplicação que permite ao utilizador anotar sobre o que está a ser apresentado (ver figura 3). Esta aplicação usa tinta digital para fazer as notas, estando disponível uma variedade de cores e tamanhos. É também disponibilizado um teclado virtual e uma borracha para apagar as anotações. Desta forma é possível ao apresentador efectuar anotações sobre a apresentação, as quais poderão ser depois disponibilizadas aos estudantes. Para isso, o apresentador tem ao seu dispor a possibilidade de guardar as anotações em memória e depois salvaguardá-las em ficheiro.

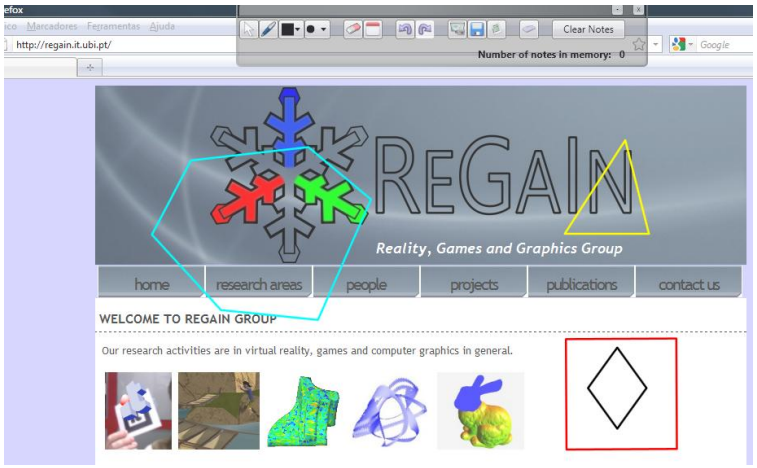


Figura 3. Aplicação iiNote.

Em vez de guardar cada nota num ficheiro de imagem através da captura do ecrã como faz o Smoothboard, o utilizador pode ir capturando as várias notas, sendo armazenadas em memória, durante a apresentação. No final, o utilizador pode exportar as notas em memória directamente para um documento XPS (XML Paper Specification) [12], ou guardá-las num ficheiro para futura edição. Deste modo, o utilizador pode partilhar rapidamente as notas tiradas durante uma apresentação.

A aplicação permite ainda suavizar o desenho e fazer o reconhecimento de formas geométricas desenhadas pelo utilizador, redesenhando-as de modo a ficarem mais perfeitas. Este reconhecedor de formas geométricas foi implementado usado a API InkAnalysis [13]. Note-se que as anotações são efectuadas à mão pelo utilizador como quem escreve ou desenha tal como num quadro a giz. Assim para notas do tipo texto, quando são reconhecidas pelo sistema, o texto é inserido em separado no documento XPS, como mostra a Figura 4.

Figura 4. Anotação exportada para um documento XPS.

3.2 Aplicação eeNote

Como complemento à aplicação iiNote, foi criada outra aplicação, designada de eeNote, que permite editar as notas capturadas anteriormente pelo utilizador. Assim é possível completar as notas tiradas durante uma apresentação, corrigindo ou apagando-as e adicionando mais notas. Esta aplicação (ver Figura 5), tal como a aplicação iiNote, permite usar tinta digital, a borracha para apagar as notas, reconhecer formas geométricas e texto, suavizar o desenho e exportar as notas para um documento XPS e/ou guardá-las em ficheiro.

Esta é uma aplicação que permite editar à posterior as anotações tiradas numa apresentação e corrigi-las e/ou melhorá-las de modo a que possam ser disponibilizadas sem erros. Desta forma, o apresentador não tem de estar demasiadamente preocupado com a perfeição das anotações que efectua durante a apresentação, pois poderá corrigir ou melhorar as mesmas posteriormente e com mais tempo.

Esta ferramenta pode ser bastante útil quando se pretende disponibilizar sem erros a apresentação juntamente com as anotações e o mais completa possível.

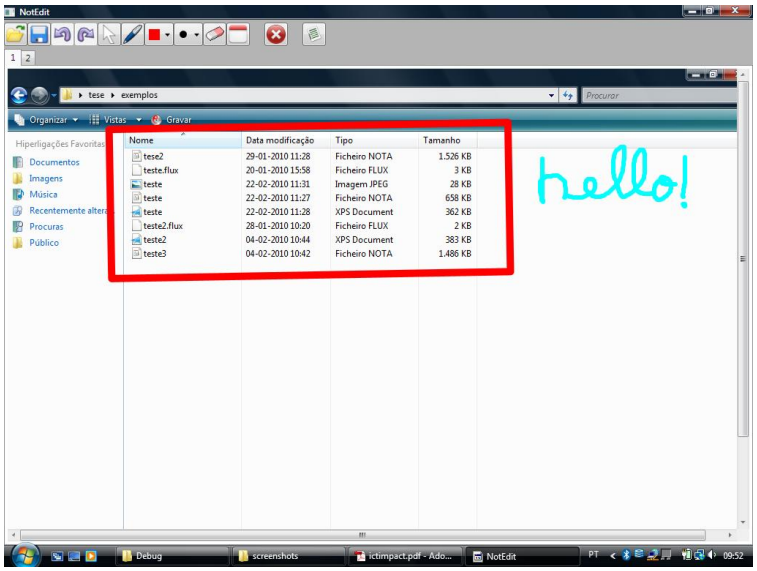


Figura 5. Exemplo da aplicação eeNote.

3.3 Aplicação iiProgramming

A outra aplicação desenvolvida serve de suporte às aulas de Introdução à Programação. Esta é uma aplicação visual que permite ao professor criar, através de um fluxograma o algoritmo em pseudo-código e o correspondente código fonte em linguagem C (ver Figura 6). Para isso, o professor escolhe a instrução pretendida de uma lista de instruções pré-definidas e, através de *drag-and-drop* coloca a instrução na posição desejada do algoritmo.

Como o objectivo da aplicação é o apoio ao ensino, esta apenas suporta instruções genéricas e simples tais como: ler do teclado; escrever no ecrã; declarar variáveis; usar ciclos While, Do-While e For; efectuar testes If-Then, IF-Then-Else e Switch-Case; e suporta ainda funções definidas pelo utilizador. A aplicação permite apenas o uso de quatro tipos de dados: int, float, char e char[].

A aplicação para além de gerar o pseudo-código e código fonte associado ao fluxograma criado permite ainda guardar o fluxograma em ficheiro para poder ser distribuído aos alunos (ver Figura 7). No entanto, a aplicação não faz verificação de instruções, ou seja, não verifica se uma instrução inserida pelo utilizador está bem definida como, por exemplo se é feita uma operação entre variáveis de tipos diferentes. Actualmente a aplicação apenas cria o código correspondente às instruções definidas pelo utilizador não fazendo qualquer verificação de sintaxe.

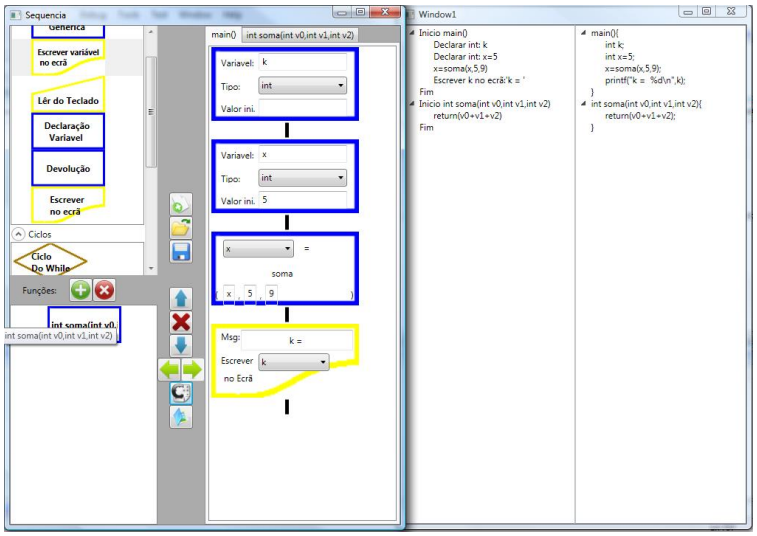


Figura 6. Exemplo da aplicação iiProgramming.

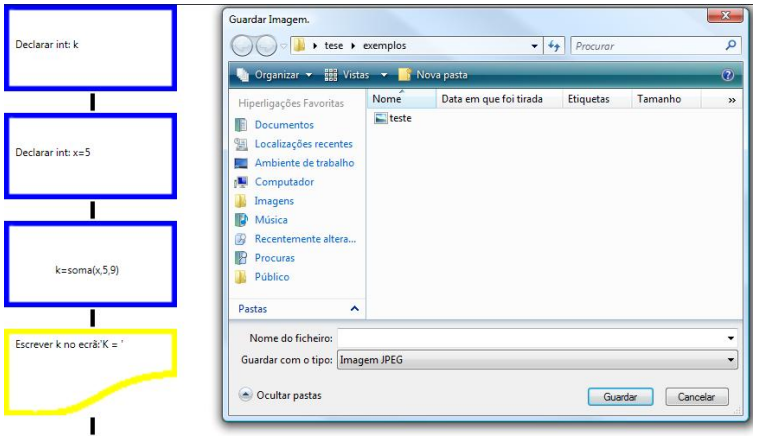


Figura 7. Fluxograma simplificado.

4 Conclusões e trabalho futuro

Durante algumas demonstrações do sistema desenvolvido, efectuadas a várias turmas de escolas secundárias, notou-se que mesmo aquelas que já têm acesso a quadros interactivos, e por isso já estão mais familiarizados com este tipo de tecnologias, tanto alunos como professores, mostraram-se entusiasmados com o funcionamento das aplicações desenvolvidas. No entanto, no futuro é preciso

ainda efectuar alguns testes de usabilidade do sistema de forma a validá-lo com um maior número de utilizadores.

Podemos concluir ainda que, pelo facto da aplicação iiNote guardar as notas em memória de um modo simples e rápido, leva a que o tempo que o apresentador perde a armazenar essas notas seja bastante pequeno, maximizando por isso o tempo de duração da apresentação.

Além disso, a possibilidade de editar as notas posteriormente leva a que o professor/apresentador não esteja preocupado se as notas estão perfeitas, permitindo ao professor/apresentador corrigir as notas depois sem limitações de tempo. No entanto, pretende-se que a aplicação eeNote venha também a permitir a introdução de texto através do teclado, de modo a poder complementar ainda mais as notas.

Em relação à aplicação iiProgramming, esta ainda apresenta algumas limitações, pois está bastante dependente dos dados introduzidos pelo utilizador de modo a produzir código sem erros. No entanto, pretende-se no futuro implementar um analisador de instruções de modo a assistir o utilizador a gerar código correcto.

O uso de um segundo Wiimote, em conjunto com o Wiimote Whiteboard para permitir o controlo remoto do computador, cancela quase completamente a dependência que o apresentador tem do rato e do teclado. Isto permite ao apresentador estar perto do quadro onde, além de o poder usar sem limitações, pode também controlar à distância o que quer ver projectado no quadro (e.g., PowerPoint, imagens, vídeos, etc.). No entanto, devido ao número limitado de botões do Wiimote, não foi possível implementar algumas funcionalidades tais como, o scroll do rato ou as teclas Backspace, Delete, Page Up e Page Down. Por isso, no futuro esperamos poder desenvolver o nosso próprio dispositivo para controlar o computador à distância e, então, eliminar as limitações atrás referidas.

Referências

1. Council of the European Union: Official Journal of the European Communities C 142/1. Brussels, 2002.
2. Balanskat, A., Blamire, R., Kefala, S.: The ICT Impact Report: A review of studies of ICT impact on schools in Europe. European Schoolnet, 2006.
3. Wikipedia: Interactive whiteboard. Disponível em `http://en.wikipedia.org/wiki/Interactive_whiteboard`, consultado em Março de 2010.
4. Wikipedia: Smart Board. Disponível em `http://en.wikipedia.org/wiki/Smart_Board`, consultado em Março de 2010.
5. Silva, M., Reis, L., Sousa, A., Faria, B., Costa, A.: iiBOARD, Development of a Low-Cost Interactive Whiteboard using the Wiimote Controller. International Conference on Computer Graphics Theory and Applications, Lisboa, 337–344 (2009).
6. Luidia Inc.: Interactive Whiteboard - Enhance Classroom Communications. Disponível em `http://www.luidia.com/products/ebeam-edge-for-education-page.html`, consultado em Abril de 2010.
7. Laxmidas, D.: Aulas 2.0. Exame Informática nº 177, Março de 2010.

8. Lee, J.: Wii Projects. Disponível em http://johnnylee.net/projects/wii/, consultado em Março de 2010.
9. Lino, F., Dias, P., Oliveira, A., Santos, B.: Comparação de Dispositivos de Interacção em Ambientes de Realidade Virtual: Desenvolvimento de um Setup Experimental e Estudos com Utilizadores. 17º Encontro Português de Computação Gráfica, Covilhã, 175–183 (2009).
10. SmoothboardWiki.: Mount and position theWiimote. Disponível em http://www.boonjin.com/smoothboard/index.php?title=Mount_and_position_the_Wiimote, consultado em Março de 2010.
11. Smoothboard.net.: Smoothboard - The Wiimote Whiteboard. Disponível em http://www.smoothboard.net/, consultado em Março de 2010.
12. Microsoft.com.: Explore the features: XPS documents. Disponível em http://www.microsoft.com/windows/windows-vista/features/xps.aspx, consultado em Março de 2010.
13. Egger, M.: Analyze This: Find New Meaning In Your Ink With Tablet PC APIs In Windows Vista. MSDN Magazine, Março de 2006.
14. Wikipedia. Windows Presentation Foundation. Disponível em http://en.wikipedia.org/wiki/Windows_Presentation_Foundation, consultado em Setembro de 2009.

WAACT - Widget Augmentative and Alternative Communication Toolkit

Gonçalo Fontes[1] , Salvador Abreu[2],

[1] LabSI{2} – ESTIG, Instituto Politecnico de Beja, Portugal
goncalo.fontes@ipbeja.pt
[2] Universidade de Évora e CENTRIA FCT/UNL, Portugal
spa@di.uevora.pt

Sumário. Neste artigo descreve-se uma proposta de implementação de uma plataforma de desenvolvimento de sistemas de Comunicação Aumentativa e Alternativa para programadores, com o objectivo de melhorar a produtividade e diminuir os tempos dispendidos na implementação deste tipo de soluções. Esta proposta assenta numa estrutura composta por widgets configuráveis por código, integráveis em novas aplicações, numa filosofia de reaproveitamento de objectos e funcionalidades. Esta plataforma pretende ainda dar flexibilidade aos programadores, através da possibilidade de introdução de novas funcionalidades e widgets. A implementação em tecnologias open source independentes da plataforma, permitirá utilizar os objectos deste toolkit em vários sistemas operativos.

Abstract. In this article we describe an implementation proposal for an Augmentative and Alternative Communication Framework for developers, with the objective of improves the productivity and reduces the implementation times for these types of solutions. This proposal is based on a customized widgets structure that can be integrated in new applications, with the objective of reuse common features of these applications. This framework intends to provide flexibility to programmers giving them the possibility of introduce new functionalities and widgets. The implementation based on open-source technologies, platform independent, allows the use of this toolkit in several different operating systems.

Palavras-chave: Framework, ToolKit, Comunicação Aumentativa e Alternativa, Pessoas com necessidades especiais, Tecnologias de Apoio.

1 Introdução

A comunicação é uma prática e uma necessidade fundamental para todos os seres humanos. No entanto, por diversos motivos, muitos de nós sofrem de condições físicas que tornam a comunicação tradicional difícil ou até mesmo impossível.

Certas perturbações sensoriais, cognitivas ou motoras podem comprometer total ou parcialmente as capacidades comunicativas humanas. As estimativas apontam para que cerca de 10% da população mundial seja portadora de um qualquer tipo de

deficiência, sendo que nesse grupo, uma percentagem bastante significativa é afectada por deficiências ao nível da comunicação [1].

Nestas circunstâncias pode-se recorrer à Comunicação Aumentativa e Alternativa (CAA).

Segundo a American Speech-Language-Hearing Association (ASHA), esta área de prática clínica tenta compensar de forma temporária ou permanente incapacidades de comunicação por parte de pessoas com dificuldades ao nível da fala ou escrita [2]. Ainda segundo a ASHA um sistema de CAA é, "um grupo integrado de componentes, incluindo símbolos, ajudas, estratégias e técnicas usadas por indivíduos para melhorar a comunicação" [1].

Os sistemas de informação e as tecnologias actuais podem ser um importante auxílio para os sistemas de CAA, tendo sido desenvolvidos nos últimos anos esforços nesse sentido com o desenvolvimento e a implementação de diversas soluções. No entanto, grande parte das soluções até agora desenvolvidas, não tem em consideração qualquer integração com outros sistemas, não sendo assim aproveitado o trabalho já desenvolvido e o conhecimento já adquirido de forma a fazer mais e melhor.

No desenvolvimento de soluções deste tipo, é importante a incorporação de todo o discurso existente, vocalizações, gestos e sempre que necessário o recurso a elementos externos de apoio à comunicação (e.g. tabela com letras e frases). Neste sentido têm vindo a ser desenvolvidos diversos sistemas tecnológicos que auxiliam a composição e transmissão de mensagens escritas ou faladas.

O Laboratório de Sistemas de Informação e Interactividade (LabSI[2]), em conjunto com o Centro de Paralisia Cerebral de Beja (CPCB) e o Instituto de Engenharia de Sistemas e Computadores Investigação e Desenvolvimento em Lisboa (INESC-ID), têm vindo a desenvolver algumas ferramentas de apoio à CAA, entre as quais se destaca o "Eugénio – O Génio das Palavras" [3].

Este sistema é uma ferramenta de apoio à escrita de textos em Português Europeu que recorre a modelos estatísticos baseados em n-gramas[4] para sugerir um conjunto de palavras prováveis na sequência do texto já escrito. O Eugénio funciona no ambiente MS Windows e apoia a escrita de mensagens em qualquer aplicação deste sistema operativo.

Para pessoas incapacitadas de utilizar um teclado de computador, o sistema dispõe de um teclado de ecrã[1]. A selecção destes elementos pode ser efectuada através de métodos de acesso directo que utilizam, por exemplo, um dispositivo de ponteiro (e.g. rato, caneta, luva virtual, entre outros.), ou métodos de acesso indirecto que recorrem apenas a um ou dois interruptores. Para o reforço da interacção com o utilizador foi incorporado no sistema um agente de interface e um sintetizador de fala, podendo estes componentes da interface do sistema ser adaptados às necessidades particulares de cada pessoa.

No desenvolvimento de outros sistemas de apoio à CAA, como é o caso do Caderno Escolar – Electrónico (CE-e)[5] tem-se verificado a necessidade de reutilização de componentes de software já desenvolvidos para outros sistemas. Por

[1] Componente que apresenta uma matriz contendo os vários caracteres disponíveis para a composição de mensagens.

exemplo, no CE-e, que pretende ser uma alternativa digital aos tradicionais cadernos de papel para alunos com necessidades especiais, verificou-se a utilidade de incorporação de algumas funcionalidades já desenvolvidas para o Eugénio, como a predição de palavras ou o teclado de ecrã. Além disso, se estas ferramentas adoptarem uma estrutura modular que facilite a substituição dos seus componentes, então poder-se-ão experimentar e avaliar de forma mais eficaz novas abordagens a determinadas técnicas de CAA, como é o caso da predição de palavras.

Contudo, a integração de componentes entre as duas aplicações anteriormente descritas não é tarefa fácil, visto que tais sistemas não foram desenvolvidos de forma modular e nem sequer na mesma tecnologia.

Neste contexto, este artigo propõe uma plataforma de desenvolvimento transversal ao sistema operativo que permita a reutilização de componentes de software, numa linguagem multiparadigma e de uso geral.

A nossa solução, que se encontra neste momento em desenvolvimento, é implementada numa arquitectura modular que permite a reutilização de variados widgets ou a criação de novos componentes, podendo estes ser usados isoladamente, num novo projecto ou ainda integrados num projecto existente. Devido ainda ao middleware utilizado e a um módulo de gestão de eventos criado, garantimos total liberdade na compilação e execução dos projectos para os sistemas operativos mais comuns do mercado.

Por fim, devido ao facto das tecnologias utilizadas serem abrangidas pelo licenciamento LGPL, é-nos garantida total liberdade no desenvolvimento e disponibilização dos componentes.

De forma a introduzir o trabalho até agora desenvolvido, na secção 2 apresentamos uma revisão do estado da arte, sendo na secção 3 apresentado e descrito o sistema. Na secção 4 será explicada a avaliação e o trabalho relacionado, e por fim, na secção 5, concluímos e propomos direcções para trabalho futuro.

2 Estado da Arte

É comum, no que diz respeito às tecnologias de informação, que no desenvolvimento de um software ou sistema, o público portador de um qualquer tipo de deficiência cognitiva ou motora, acabe por não ser contemplado pelos requisitos das várias fases do projecto. Duas das razões para esta situação prendem-se com a especificidade que os sistemas devem ter neste tipo de casos e a variedade de situações entre cada utilizador e o seu estado.

No entanto, o desenvolvimento de software de apoio à CAA é imprescindível para as pessoas que conseguiram ganhar parcial ou total autonomia graças as tecnologias, bem como para futuros utilizadores.

2.1 Comunicação Aumentativa e Alternativa

Um dos principais objectivos desta área, é fornecer a ajuda necessárias as pessoas incapazes de satisfazer as suas necessidades diárias de comunicação através de meios convencionais.

Ao contrário do que normalmente é sugerido, a Comunicação Aumentativa e Alternativa abrange todas as formas de comunicação, não sendo esta apenas um exclusivo da forma oral, sendo assim usada para a expressão de pensamentos, necessidades, vontades ou ideias. Todos usamos CAA, mesmo que inconscientemente, quando fazemos expressões faciais ou gestos, usamos símbolos ou imagens, ou ainda quando escrevemos. Todas estas utilizações de CAA permitem-nos usufruir de uma prática fundamental, a comunicação.

As pessoas com graves dificuldades ao nível da fala ou expressão têm nestes métodos de comunicação, uma forma de complementar o seu discurso existente, ou, se este for inexistente, uma forma de o substituir. Para se exprimir, estas pessoas dispõem de diversas ajudas aumentativas, como os quadros de comunicação por imagens ou símbolos, ou ainda equipamentos electrónicos, no entanto, mesmo com este tipo ajudas, é importante que os seus utilizadores nunca deixem de se exprimir de uma forma natural, se o conseguirem, devendo estes métodos ou mecanismos ser apenas uma melhoria na sua comunicação.

Contudo, se estas ajudas permitem uma melhoria significativa da comunicação dos seus utilizadores, estas também apresentam alguns problemas, tais como o facto de a sua utilização ser bastante lenta comparativamente com os métodos de expressão mais utilizados, a fala ou escrita. Este é um dos principais problemas ligados ao desenvolvimento de soluções de auxílio a comunicação, sendo que os dois valores mais importantes expressados por pessoas que utilizam este tipo de sistemas são: (i) Dizer exactamente o que querem dizer; (ii) Dizê-lo o mais depressa que consigam [6].

2.2 Software de apoio a CAA

A introdução dos sistemas de informação e das tecnologias actuais vieram melhorar significativamente a eficácia dos sistemas de CAA.

Como já foi referido, ao logo dos últimos anos têm sido desenvolvidos muitos esforços da construção de soluções tecnológicas com o intuito de auxiliar os utilizadores com dificuldades comunicativas.

A grande maioria dos softwares de CAA desenvolvidos tem por base, de uma forma ou de outra, um teclado de ecrã, tanto na sua forma tradicional funcionando nos tradicionais computadores de secretária ou portáteis, como em interfaces físicas adaptadas (dispositivos do tipo "handheld"), sendo os mesmos utilizados para a obtenção de frases, podendo estas ser formadas por letras, palavras ou por sequências de símbolos pictóricos. Um exemplo deste tipo de sistemas é o Eugénio (Fig. 1.) [3]. Esta ferramenta que dispõe de uma funcionalidade de predição recorre a modelos estatísticos baseados em n-gramas[4] para sugerir um conjunto de palavras prováveis na sequência do texto.

A ideia por detrás das n-gramas ou modelo de Markov é prever a probabilidade da letra seguinte num determinado texto. Para isso são examinadas amostras de textos denominadas de textos de treino que, com base numa estimativa de parecenças, irão identificar a letra com maior probabilidade possível de ser a seguinte.

Para além da predição de palavras, o Eugénio dispõe de um sistema de varrimento da aplicação que permite aos utilizadores que tenham deficiências motoras, estando por isso impossibilitados de usar os tradicionais mecanismos de I/O, como é o caso dos dispositivos apontadores, usar métodos que recorrem a um ou dois interruptores, para que o utilizador não tenha que movimentar os membros podendo apenas seleccionar a opção desejada.

Como já foi também referido, um dos principais requisitos para os sistemas de CAA é o facto de o utilizador poder ser rápido na escrita da mensagem que pretende comunicar. Assim, para além dos mecanismos já descritos, o Eugénio dispõe ainda de um mecanismo de extensão de abreviaturas que permite ao utilizador escrever palavras ou frases digitando apenas a sua abreviatura. Esta funcionalidade recorre a um dicionário de abreviaturas predefinido, podendo este ser configurado para um incremento dos termos, permitindo uma melhor adaptação ao seu utilizador.

Fig. 1. Interface do Eugénio apresentando o teclado de ecrã e a predição de palavras e interface do Talkactive apresentando o teclado de símbolos pictóricos.

Isolando todas estas funcionalidades, podem-se extrair componentes dos mais diversos tipos, como a predição de palavras, o teclado de ecrã, o varrimento e a expansão de abreviaturas. Todos estes componentes que trabalham em conjunto foram implementados sobre uma estrutura hierárquica de classes. Todos eles seriam de uma grande utilidade em sistemas similares.

Para além do Eugénio, existem algumas outras ferramentas de CAA que permitem a expressão de palavras ou frases, quer de uma forma escrita, quer de uma forma falada, por intermédio de sintetizadores de voz. Um desses casos é o Talkactive (Fig. 1.)[7] que recorre a modelos de High Frequency Vocabulary ou Core Vocabulary: Este tipo de modelo define que um número relativamente pequeno de palavras pode

constituir a grande maioria do que é dito em comunicações normais. Com algumas centenas de palavras uma pessoa pode dizer cerca de 80% de tudo o que é necessário em comunicações diárias [8].

Este é um sistema de utilização simples que recorre a símbolos pictóricos para identificar palavras, que conjugadas entre si permitem formar as frases que o utilizador pretende comunicar.

Assim, o utilizador apenas precisa de seleccionar as imagens representativas do que este pretende transmitir a outro interveniente, tendo a vantagem de num único click poder transmitir uma palavra, frase ou ideia.

Devido em grande parte a sua simplicidade, estes sistemas de símbolos ganharam grande popularidade em vários países, sendo usados com grande sucesso por pessoas portadoras de paralisia cerebral [9][10].

Contudo, o Talkactive também possui claras desvantagens, como o facto de, devido a sua extensão, não ser possível disponibilizar num único ecrã todo o vocabulário de utilização diária ou de não estar disponível em português.

No entanto, à semelhança do que se passa com o Eugénio esta aplicação também é composta por diversos componentes distintos, tais como o teclado, o corpus de símbolos pictóricos e um editor de texto, que poderiam por sua vez também ser de grande utilidade em aplicações do mesmo género.

Por outro lado, estes dois sistemas acabam por se revelar bastante semelhantes ao nível da sua estrutura gráfica, bem como, ao nível de algumas das suas funcionalidades, sendo ambos baseados numa grelha de teclas permitindo a escrita de palavras e frases.

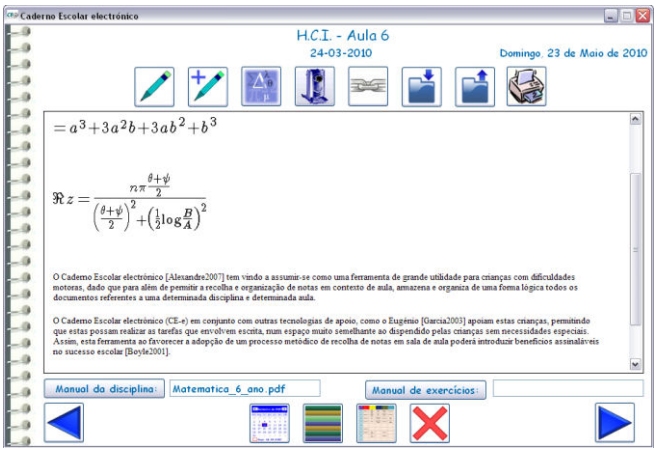


Fig. 2. Interface principal do notetaking do CE-e, apresentando a inserção de texto e de formulas matemáticas.

Outro software de apoio a pessoas com necessidades especiais, mas desta vez essencialmente direccionado para estudantes, é o Caderno Escolar electrónico (CE-e) [5]. Este sistema que pretende ser um substituto dos tradicionais cadernos escolares em papel, disponibiliza aos seus utilizadores uma interface e variadas funcionalidades de notetaking organizado, permitindo a escrita de texto, inserção de imagens e

hiperligações, upload de ficheiros e aplicação de fórmulas matemáticas, entre outras (Fig. 2.).

No entanto, ao contrário dos sistemas anteriores este não possui qualquer ferramenta de apoio ou aceleração da escrita, como o preditor de palavras, o teclado de ecrã com varrimento ou ainda a grelha de símbolos pictóricos, ferramentas estas, de grande importância para aumento da produtividade deste tipo de utilizadores.

2.3 Sistemas com Paradigmas Semelhantes

A ideia da criação de uma plataforma de auxílio ao desenvolvimento de aplicações de CAA não é propriamente nova, tendo sido nos anos 90 desenvolvidos esforços nesse sentido por parte do Consórcio Comspec. Este consórcio que reuniu uma equipa multidisciplinar constituída por educadores, engenheiros e programadores provenientes de diversos países europeus [11] tinha por objectivo a criação de uma plataforma que permitisse plena transversalidade entre quatro tipos de utilizadores (programadores, integradores de sistema, facilitadores e utilizadores finais).

Contudo, devido a necessidade de garantir o cumprimento dos requisitos necessários a todos estes utilizadores, acabaram por serem impostas grandes limitações na interface bem como na configuração de componentes, o que levou ao esquecimento do projecto, alguns anos mais tarde.

O projecto Ulysses tentou ser um pouco menos ambicioso que o Comspec tendo sido definidos apenas três tipos de utilizadores principais (programadores, integradores e utilizadores finais), no entanto o desenvolvimento do sistema acabou por não conhecer grandes avanços, tendo acabado por ser abandonado [14].

3 Apresentação e Descrição do Sistema

Como foi dito na secção 2, a maior parte dos sistemas de apoio à CAA foram desenvolvidos de forma monolítica, sendo a sua implementação, a sua alteração ou integração noutros sistemas bastante difícil.

Este problema acaba por ter diversas origens, tais como o desenvolvimento de aplicações proprietárias, a falta de estrutura para reaproveitamento de código, ou ainda, a utilização em diferentes estruturas ou linguagens durante a sua implementação. São estes os problemas que este projecto pretende resolver.

Para isso, encontra-se em fase de estudo e codificação uma plataforma de desenvolvimento para programadores, denominada de Widget Augmentative and Alternative Communication Toolkit (WAACT), que lhes permitirá criar projectos de ferramentas de CAA, utilizando uma variedade de componentes gráficos predefinidos e configuráveis comuns neste tipo de aplicações, permitindo assim aumentar a produtividade no desenvolvimento, bem como experimentar e avaliar novas abordagens a estas aplicações durante a investigação.

3.1 Framework ou Toolkit

Analisando as suas características e estrutura podemos classificar esta ferramenta como Toolkit ou como Framework.

Existem varias definições para ambas as estruturas entre as quais se destaca, para os Frameworks, a de Ralph E. Johnson que diz que estes "são a reutilização da totalidade ou de parte do desenho de um sistema que é representado por um conjunto *de classes abstractas e pela forma como as suas instâncias interagem"*[13].

Ainda o mesmo autor diz que *"um* Framework é um esqueleto de uma aplicação *que pode ser configurada por um programador"*.

Assim, quando se utiliza um Framework o corpo da aplicação está implementado, permitindo a sua reutilização, devendo apenas ser particularizadas as chamadas às funções, o que reduz significativamente as decisões de concepção, ao contrário do que é feito em sistemas como os Toolkits ou livrarias partilhadas(*dll's*), em que é codificado o corpo das aplicações e apenas se faz a chamada ao código que se quer reutilizar [14].

Não sendo as duas definições anteriormente descritas semelhantes, também não são contraditórias, acabando até por se completar uma a outra. Enquanto a primeira define a estrutura de um Framework a segunda define o seu objectivo.

O WAACT pode assim ser considerado um Toolkit com características de Framework.

3.2 Análise e Decisões

Durante a análise deste projecto foram surgindo diversas questões, tais como, a quem seria o sistema disponibilizado e quem o poderia desenvolver.

Assim, como o principal objectivo é a uniformização da estrutura das aplicações em questão bem como a reutilização de componentes já desenvolvidos e validados, verificou-se que a melhor forma de atingir esse objectivo seria através da participação no desenvolvimento de todos aqueles que utilizarão o sistema. Para isso, foram escolhidas ferramentas e linguagens de uso geral e abrangidos por licenciamento aberto, permitindo a utilização e possível desenvolvimento de todos.

Contudo, o uso de sistemas Open Source não era de todo suficiente, tendo em conta que a maior parte dos profissionais destas áreas acabam por trabalhar em sistemas e softwares proprietários, tendo sido assim necessário a utilização de tecnologias multi-plataforma, bem como a criação de uma estrutura modular que permita um desenvolvimento contínuo e flexível.

O desenvolvimento do WAACT é feito em C++ e apoiado na User Interface Framework, QT [15], o que permite resolver alguns dos problemas atrás referidos.

Por outro lado, a necessidade de garantir a gestão dos eventos enviados por equipamentos de Input (podendo estes ser ratos, teclados, interruptores, entre outros.), processando-os e, se necessário, enviando-os para algum equipamento de Output, originou a criação de uma interface Event Manager que tem por missão tornar os módulos independentes dos eventos, permitindo o isolamento de cada Widget, podendo estes ser desenvolvidos com total independência uns dos outros (Fig. 3.).

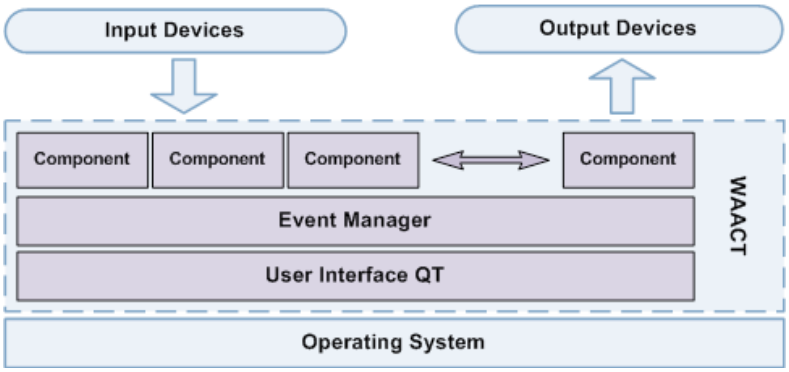


Fig. 3. Estrutura de implementação do WAACT

Sendo o QT também desenvolvido em C++, este middleware permite uma grande liberdade e interoperabilidade entre sistemas operativos, podendo assim ser desenvolvidas aplicações para as diversas plataformas existentes no mercado, com recurso à versatilidade que o paradigma da programação orientada a objectos oferece.

Assim, as aplicações desenvolvidas não só podem ser implementadas com recurso às bibliotecas standard do C++ como podem ser programadas com recurso as bibliotecas específicas do QT.

O QT dispõe ainda de um mecanismo de SLOTs e SIGNALs que se podem definir como uma alternativa às técnicas de callback. Estas são as funcionalidades centrais deste sistema e o maior aspecto diferenciador de outros Frameworks. Esta funcionalidade permite ligar funções a eventos, ou funções a outras funções, o que permite despoletar acções em qualquer tipo de evento predefinido ou em qualquer evento programado.

3.3 Utilização

Como já foi dito, foi necessária a utilização de um middleware que fosse multi-plataforma, de forma a não restringir a utilização de software desenvolvido a um determinado sistema, tendo sido o QT a plataforma escolhida.

Esta escolha permite alguma flexibilidade ao programador, pois como também já foi referido, este poderá desenvolver utilizando varias tecnologias. Contudo, pensamos ser importante proporcionar aos utilizadores do WAACT uma metodologia de implementação mais direccionada e que não obrigue a utilização de todas as tecnologias disponíveis na plataforma.

Assim, numa óptica de programação orientada a objectos, foram criadas, para além dos widgets, funções que permitissem absorver as bibliotecas do QT, para que o programador apenas programe em C++ com recurso ao WAACT, libertando o utilizador de mais uma tecnologia.

Exemplo de código na programação do WAACT

```
W_KeyBoardKey *k = new W_KeyBoardKey();
```

```
k->keySized("B", "b", 50, 50);
W_TextPad *t = new W_TextPad();
t->textPad();
EventManager *e = new EventManager();
e->clickedKeyToTextpad(k, t);
```

4 Avaliação e Trabalho Relacionado

Numa primeira fase do projecto, foi importante avaliar a real mais-valia que poderia ser proporcionada por um toolkit deste tipo no desenvolvimento de software específico.

Normalmente este tipo de avaliação é feita numa fase final de implementação, no entanto, sentimos a necessidade de validar as escolhas e caminhos percorridos, tendo-nos então decido a implementar algumas ferramentas já desenvolvidas noutras tecnologias de forma a dai tirar algumas ilações.

Uma dessas ferramentas foi o CE-e, da qual implementamos um protótipo bastante simples que apenas dispunha das principais funcionalidades de notetaking.

Para além, da portabilidade entre vários sistemas operativos, tendo este protótipo sido compilado, executado e testado em Sistemas Microsoft e Linux (Fig. 4.); este teste demonstrou-nos que estas aplicações poderiam ser implementadas com um relativo baixo esforço de programação, desde que os widgets pretendidos estejam disponíveis no WAACT. Verificamos assim, que com poucas linhas de código, poderemos obter uma aplicação gráfica, mesmo que rudimentar, mas funcional.

Como já foi referido anteriormente, este projecto assentou no pressuposto da reutilização de componentes existentes noutras aplicações, bem como no teste de novas abordagens de integração dos mesmos, tendo sido introduzido no nosso protótipo um teclado de ecrã bastante semelhante ao do Eugénio (Fig. 4.).

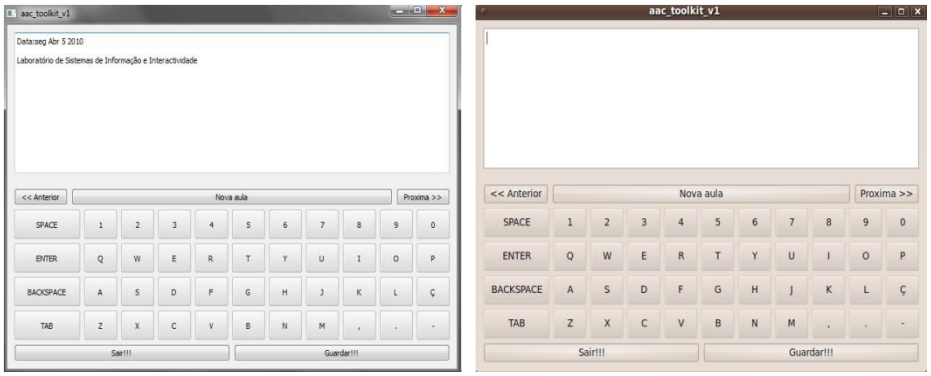


Fig. 4. Protótipo de uma aplicação desenvolvida pelo WAACT em ambiente Windows e Linux

Neste caso fizemos a integração de um teclado QWERTY (disposição adoptada pelos países ocidentais), no entanto, poderão ser utilizados ou criados outros, sendo

que, o WAACT está preparado para qualquer disposição de teclas, podendo inclusivamente ser criado um teclado de raiz.

Baseado no protótipo anterior, a integração de um teclado virtual, que é um componente comum e bastante utilizado, necessitou apenas do registo de mais duas linhas, no código já implementado.

Assim, com base nestes dados, verificamos que desde que alguns dos componentes necessários ao desenvolvimento de determinada aplicação estejam implementados no WAACT, o aumento de produtividade e eficiência é visível no desenvolvimento de soluções deste tipo.

Por outro lado, é possível garantir flexibilidade para a investigação, podendo ser integrados ou trocados diversos componentes, ou ainda para o desenvolvimento, podendo ser avaliadas novas perspectivas por parte do programador.

5 Conclusões e Trabalho Futuro

Neste artigo propusemos e descrevemos o WAACT e qual o seu objectivo, passando ainda pelas necessidades existentes ao nível de softwares CAA bem como pela história dos mesmos.

Como já foi referido este é um trabalho que se encontra em curso, estando ainda muito do desenvolvimento por efectuar, no entanto, o actual estado permite-mos verificar uma necessidade crescente na existência de uma plataforma com as características apresentadas pelo WAACT.

Actualmente dispomos de variados widgets implementados, tais como, teclados de ecrã e seus componentes, caixas rich text, botões gerais e específicos, entre outros.

Para além dos widgets propostos dispomos ainda de um módulo chamado de Event Manager que gere os eventos de input, processando-os e despoletando um evento de output, caso seja esse o objectivo, permitindo ainda o desenvolvimento de componentes dependentes ou independentes dos já existentes.

Contudo, este é um trabalho que deve ser contínuo e atento às necessidades nas áreas de CAA, devendo ainda ser implementados vários componentes que permitam completar, pelo menos para já, o leque das necessidades actuais.

Pretendemos ainda, numa fase mais avançada, disponibilizar o WAACT em regime de "Open Source", bem como uma documentação adequada que permita uma fácil utilização e expansão por parte dos programadores.

No entanto, nesta fase, podemos desde já validar o QT como uma escolha bem sucedida no que diz respeito as questões de portabilidade, bem como, no que diz respeito a qualidade gráfica fornecida no desenvolvimento de aplicações.

Agradecimentos

Nesta secção, gostaríamos de agradecer a colaboração do corpo docente do Instituto Politécnico de Beja afecto ao Laboratório de Sistemas de Informação e Interactividade (LabSI^2), pelo contributo e ideias dadas para esta plataforma.

Referencias

1. Universidade de Aveiro – Biblioteca digital, http://portal.ua.pt/bibliotecad
2. Scvcik, R., Romski. M.: Aac: More than three decades of growth and development. The ASHA Leader (2000)
3. Garcia, L.: Concepção, Implementação e Teste de um Sistema de Apoio à Comunicação Aumentativa e Alternativa para o Português Europeu. Master Thesis, Instituto Superior Técnico (2003)
4. Brown, P. F., Pietra, V. : Class-Based n-gram Models of Natural Language. Association for Computational Linguistics. Volume 18, Number 4 (1992)
5. Alexandre, L., Garcia, L., Bruno, L.: Development of an Electronic Scholar Notebook for Students with Special Needs. DSAI2007 - Vila Real (2007)
6. AAC Institute, http://www.aacinstitute.org
7. Sensory Software International, Lda., http://www.sensorysoftware.com/Talkative.html
8. Vanderheiden, G. C., Kelso, D. P.: Comparative Analysis of Fixed-Vocabulary Communication Acceleration Techniques. AAC Augmentative and Alternative Communication, 3, 192--206 (1987)
9. Besio, S., Ferlino L.: Blissymbolics Software Worldwide: from Prototypes towards Future Optimized Products. In Proceedings of the 2nd ECART Conference (1993)
10. Schlosser, R. W., Koul, R., Raghavendra, P., Lloyd, L. L.: State-of-the-Art in Blissymbolics Research: Implications for Practice. In Proceedings of the 6th ISAAC Conference, 121--123 (1994)
11. Lundälv, M.: Comspec - The Advent of an Integrated Modular Communication System. Proceedings of the Future Integrated Solutions Conference, ACE Centre, Oxford (1994)
12. Kouroupetroglou, G., Pino, A.: A New Generation of Communication Aids under the ULYSSES Component-Based Framework. The 5th International ACM SIGCAPH Conference on Assistive Technologies (2002)
13. Johnson, R. E.: Components, Frameworks, Patterns. Communications of the ACM (1997)
14. Gamma, E., Helm, R., Johnson, R., Vlissides, J. M.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994)
15. Nokia QT – Cross Platform Application and UI Framework, http://qt.nokia.com

Computação Móvel e Ubíqua

A system for coarse-grained location-based synchronisation

André Coelho[1], Hugo Ribeiro[1], Mário Silva[1], Rui José[2]

[1]Mestrado em Informática, Universidade do Minho
[2]Departamento de Sistemas de Informação, Universidade do Minho
andresilvacoelho@sapo.pt, hugomsribeiro84@gmail.com, mario_jsilva@netcabo.pt, rui@dsi.uminho.pt

Abstract. This paper describes a system for supporting coarse-grained location-based synchronisation. This type of synchronisation may occur when people need only some awareness about the location of others within the specific context of an on-going activity. We have identified a number of reference scenarios for this type of synchronisation and we have implemented and deployed a prototype to evaluate the type of support provided. The results of the evaluation suggest a good acceptance of the overall concept, indicating that this might be a valuable approach for many of the indicated scenarios, possibly replacing or complementing existing synchronisation practices.

Keywords: location-based synchronisation, synchronised activity, connectedness, reassurance, remote presence awareness, calendar system.

1 Introduction

Daily life is full of situations in which we have to synchronise our actions with other people. This is an integral part of social interaction and may occur in the context of very diverse social situations. For example, parents need to coordinate to get their kids from school, work colleagues may want to go to lunch together and friends may want to meet at their favourite place. Calendars and agendas are the primary tool for synchronising with others as they enable us to plan and anticipate synchronisation. However, they only represent the expectation that events will happen in a particular way at a particular moment. More recently, mobile phones have also become an important synchronisation tool, allowing people to make only basic arrangements, like "We will meet tonight at one of those bars", and then fine-tune the synchronisation process through the situated exchange of phone calls or SMSs. In fact, many phone calls start by some variant of the question "where are you", especially when there is some expectation that the other person might be somewhere nearby. These forms of synchronisation may reflect pragmatic needs associated with people finding each other, but they are also often just a reflection of a need for reassurance and connectedness towards other people [1].

In this work, we explore how the increasing ubiquity of mobile technologies may support new forms of synchronisation [2]. In particular, we explore the concept of synchronised activity as some type of social activity in which multiple people are

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 367–378

involved and desire to maintain a coarse-grained location-based synchronisation between each other. A synchronised activity associates calendar data with a certain physical scope and a set of participants. It provides the context in which those participants will be able to generate and receive relevant location-based notifications that will allow them to perceive how the activity is unfolding in terms of the location of the other participants. We hypothesise that this may extend the role of calendars from planning tools to situated synchronisation tools, and thus from a focus on plans to a focus on situated action as the essence of interpersonal synchronisation [3].

In this paper, we describe the study we have conducted to explore the viability of this concept and gain a more in-depth understanding about its potential and limitations. We start by reviewing systems that share similar objectives and concepts. We then analyse some of the main reference scenarios that we have identified for this type of synchronisation. These scenarios provided the basis for the identification of the requirements of our proposed systems for enabling synchronised activities, which we describe in Sec. 3. Based on these requirements, we propose an architecture to support these synchronisation models and we have created a prototype implementation of that system, both described in Sec. 4. In Sec. 5, we describe the evaluation procedures in which six users have tried the system simulating realistic social situations. The results highlight some important findings, but overall they seem to confirm the validity and opportunity of the concept of synchronised activity.

2 Related Work

Our work has some similarities with location sharing systems, like Locaccino or Google Latitude. Locaccino [4] is an application for desktop computers and mobile devices that enables users to share their location with people from their Facebook social network. Location sensing can be done through the use of Wi-Fi or GPS. This system puts great emphasis on the user's privacy. People can set themselves as undetectable whenever desired and they can express the location disclosure preferences using multiple variables, such as time spans and the locations where a group of people is allowed to know the user's location.

Google Latitude [5] is a location-aware application for both desktop and mobile devices that enables users to share their location with their friends with Google accounts. On the desktop, location sensing is achieved through IP geolocation while on mobile devices it is achieved through cellular positioning and GPS. This system, much like Locaccino, puts great emphasis on privacy, reason for which it offers varying degrees of precision in location information, according to what users have chosen to show to other users. Location information can be as precise as the exact GPS coordinates location or as vague as just the city name. The system also overrides old location data with new one so as to avoid the possibility of a user's activity being tracked, unless the user specifically tells the system to keep a history of his locations. The system allows for users to contact users with whom they share their location via Google Talk. This facilitates the possibility of users synchronising for some activity, especially if they are near each other.

The location sharing features of these systems also provide the ground for multiple forms of location-based synchronisation. However, in these systems, synchronisation comes as a by-product of the system's features and not as an integrated part of the tool's design. As a consequence, many of our targets scenarios cannot be properly supported or can only be supported with strongly negative consequence in terms of privacy. In our work, we do not intend to make users traceable all the time, our system is only meant to alert other users of the system when someone has arrived at a predetermined location in the context of some prearranged activity. The fact that location data is only used for the purpose of synchronising people within the scope of a specific activity, together with the potential anonymity of many of our scenarios, means that privacy is much easier to handle in our system than it is within any general purpose location sharing system.

From the perspective of supporting structured awareness about the activity of others, our system also has similarities with several types of ambient display systems. The Whereabouts Clock [6] is a system composed of an ambient display tied to a computer/SMS gateway and a mobile application. The ambient display works as a situated awareness device enabling onlookers to have a persistent, dynamic and at-a-glance view of other peoples' whereabouts. For this purpose the researchers used the clock design metaphor, divided into three portions, each indicating a user's presence in a different location, "in the building", "at home" and "out". Location sensing is achieved through the identification of GSM cells in the user's current vicinity and the different locations indicated by the display must be registered once in the mobile application. In addition users can also broadcast their activity, choosing from a specific list. This system enabled it's users to feel imbued with a sense of remote presence awareness and connectedness.

The key difference to our work is that the whereabouts clock is designed to stay in the same location, the home, and to inform the people that are in that space. Our work is very different in this respect, its purpose is to make such information available to a user anytime and anywhere, in essence, empowering users by making the information mobile. In spite of these differences, the study behind the Whereabouts Clock still allowed us to extract valuable insight, namely the need for the users of the system to understand the context of the information they receive about others and to be able adapt the system to suit their needs.

HomeNote [7] is a system that consists of a software application installed on a tablet PC which is used as a situated display in the houses of families chosen to test the application. The display can receive SMS messages and users can write notes by hand using the tablet PC's stylus. HomeNote aims to exploit the potential and value of person-to-place communication, as opposed to person-to-person communication, in a family environment. With this they aimed to extend their comprehension on the types of communication interactions that are carried on in a family environment and develop support for remote and local situated messaging. The system was regularly used for purposes of synchronisation in the context of an activity. Over the course of the study, tests showed that there were seven types of messages that were common amongst all the households where the system was tested. From those messages we would like to call attention to the following ones: Call for Action, Awareness and Reassurance, Social Touch and Reminders. These are the types of messages whose

content and social implications replicate the type of human interaction that our system intends to support and that are most relevant in the scope of synchronisation between people.

The authors conclude that by paying attention to the considerations of some mundane household technologies it is possible to support existing practices and also to create new forms of communication. This is an objective our works share with HomeNote, but on a different perspective. While HomeNote aims to explore people-to-place messaging, our system calls for a more persons-to-persons background interaction. Analysis of this project leads us to believe that, when deploying our application, there is a need to collect information about the extension, quality and diversity of the types of interactions that our application enables.

3 Reference Scenarios

In this section, we present a set of reference scenarios that demonstrate possible uses of our system and which have also been used as a basis for requirement identification.

"Let us meet here in roughly one hour". This is the scenario where a group of people arrives somewhere and then separates for some time, while doing separate activities. For example, a family may arrive together at a shopping centre. While one of the family members goes to the supermarket, the others will be visiting some local shops. They intend to meet at the end, although they do not know exactly who is going to take longer. Another example may be a tourist bus dropping tourists at a museum. The passengers are expected to be back to the bus after finishing their visit, but the duration of the visit is variable. In this scenario the synchronisation activity is one that is truly very common in everyday life. A group of people separates and agrees to meet at roughly the same time in a designated spot knowing that the subsequent activity is bound by the arrival of all the elements.

"Who is already there, who is arriving". In this scenario a store or company organizes a flash mob at some location. They intend to gather a certain number of people in that location, for that effect they might offer some sort of reward for showing up. People adhering to the activity are interested in knowing how many people have shown up already. Another example for this reference scenario is a dinner party. A group of people wanting to get together for dinner, possibly at a restaurant are interested in knowing who has and who has not yet arrived. In such a scenario, synchronisation happens for the effect of gathering multiple people around an activity at a designated location. Synchronisation information here has the role of informing people about activity attendance however, depending on the social context of the activity, the content of such information could come in different forms, due to privacy issues.

"Your ride is arriving". Two co-workers go to work together in the same car. One of the co-workers offers to pick the other one up at his house, at a specific time.

The person being picked up finds it useful to know whether or not his colleague is close to the pick-up point, so he is better able to time his arrival and avoid spending unnecessary time waiting on the street. Synchronisation in this scenario happens for the purpose of sharing a resource. Information needed for the purpose of synchronisation is more vital to one of the interested parties involved than to the other because one depends on the actions of the other in order to achieve his goal.

"Yes, he already took care of that". This scenario is typical among family members. The heads of the household always need to be in synch to coordinate their efforts with numerous tasks, picking up the children from school, picking up the laundry from the dry cleaner, grocery shopping, etc. As such there is a need to know how things are and who has done what. Synchronisation in this scenario occurs around an activity that benefits more than one person, but can be carried out by a single individual. Synchronisation comes into play because of the fact that other individuals interested in the outcome of the activity feel interest in getting feedback relative to the activity's status, in order to be reassured that things are going along as planned. For instance if one of the parents picks up the children from school, the other parent will feel a need to know when that happens and if everything goes along well.

4 System support for coarse-grained location systems

In this section, we describe the platform that we created to support coarse-grained location-based synchronisation.

4.1 Requirements

From the analysis of the previous scenarios we were able to identify the following list of requirements:

- Activity support is bounded by a temporal context in which it is to happen.
- Activity support is tied to the existence of a geographic scope associated with each activity.
- Activities must support the involvement of multiple people.
- The system must enable users to activate/deactivate synchronisation functionalities regarding an activity at a time of their choosing.
- The system must act as mediator because people might not know each other and they might not know of each others' whereabouts, but they must still be able to synchronise in the context of an activity.

Other requirements are tied to details such as configuration parameters and privacy. Depending on the social context of the activity, users may desire to enforce different privacy policies regarding identity disclosure.

4.2 Architecture

The architecture we envisioned for our system, represented in Fig. 1, is composed of three distinct entities: a mobile application running on a smartphone with internet connectivity and GPS, a server that handles all notifications to and from users of the mobile application and a shared calendar system, where activities can be specified using common mechanisms to create events in the calendar.

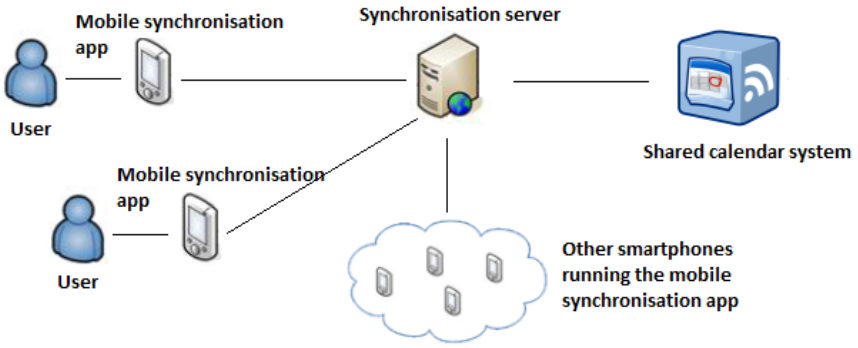


Figure 1. System architecture.

4.2.1 Mobile synchronisation application

The mobile synchronisation application is the primary point of entry into the system allowing users to create and manage activities while on the move. The mobile application supports several activity creation models. Users can create an activity while: being physically present at the site; not being present at the site but knowing the location's coordinates beforehand; or, creating an activity without location coordinates and adding them at a later time. Regarding activities the application supports different privacy policies chosen by the user for each activity; these policies will affect issues like identity disclosure. The mobile application is also responsible for warning the server when a user enters the physical region that was associated to an on-going activity. This will cause the server to generate notifications for all participants relevant to the activity and these notifications will be delivered to them via the application in their mobile devices, which effectively makes the application the endpoint for server notifications.

4.2.2 Shared calendar system

The shared calendar system (e.g. Google Calendar) provides an alternative entry point for calendar functionality and participant invitation, enabling people to create activities using a familiar interface. The participants invited to the calendar event will also become the participants in the synchronised activity The only difference for common calendar events is the possibility to encode coordinates in the calendar event and the need to include the system's own e-mail address in the invited list to make the system aware of this new activity.

4.2.3 Synchronisation server

The server is the part of the system responsible for receiving user notifications related to activities and generating and forwarding the appropriate notifications to other users in the context of said activities and according to the privacy policies appropriate for each activity type. When invited to a calendar event, the synchronisation server will interface with the shared calendar system and download activity data, such as participants list, location, start time, end time and type of activity. It will then manage the necessary notifications to the mobile synchronisation applications.

4.2.4 System operation

When someone creates a new activity, from either the mobile application or the shared calendar system, the indicated participants will be notified through their mobile applications and they can either accept or deny participation in the activity. This allows the synchronisation server to keep track of which guests have accepted or not to participate in an activity, and it allows it to manage activities for the purpose of issuing notifications to users.

When an activity's start time has been reached, the server will start accepting communication from mobile devices regarding that activity. As users enter the geographic region defined for the activity, they will be notified of this occurrence on their devices and the devices will notify the synchronisation server of user arrival. When this happens the synchronisation server will issue notifications to the mobile devices of participants informing them that a user has arrived. Depending on the privacy policy the notification can feature the identity of the person that has arrived.

Different activity types cause different notifications to be generated and with different frequency, for instance, if the activity is a flash mob, the server will notify users of arrivals at a frequency of X arrivals, so as to not annoy users with too many notifications.

4.3 Implementation

To support the evaluation of the key concepts proposed in this work, we have implemented a simplified version of our coarse-grained location-based personal synchronisation system. Based on the technical requirements we had for the mobile application, we have implemented that part of the system using the Android platform. This choice is tied to several factors, the main one being that the LBS (location-based services) API seemed very strong and provides an easy way to obtain the desired behaviour for our application in what regards user presence detection in a specified geographical region. The fact that the Android OS allows us to run applications in the background and its power management features, namely the fact that applications and GPS still work while the phone is on standby were also critical to this choice. Other factors that drew us to this platform were: the familiarity with Java and the general tidy, regulated and balanced feel of the programming model as a whole, which we feel results in applications being more suited to a mobile environment's requirements.

This implementation works as a standalone application that is designed to be tested by one individual carrying an Android device. All behaviour related to other users, in the context of an activity, is simulated by the prototype via notifications pertaining to alterations in the system caused by a user's actions, like arriving at the rendezvous point.

5 Evaluation

The overall objective of our evaluation was to gain some insight into the viability of the concept of coarse-grained synchronisation and inferring its potential as a method for replacing or complementing existing synchronisation practices. Within this broader objective, we also intended to assess more specific characteristics of our implementation, such as:

- Determine user's ease of use and learning of the application's interface;
- Determine whether the users were able to ascertain the general state of the activities he is involved in, as well as the repercussions of his actions for the system and for the other users of the system, from the perspective of what it means to be in synch with other people in the context of an activity;
- Determine if the system's feedback was appropriate and useful to users in regards of achieving the objective of synchronising with other people.

5.1 Methodology

The methodology for this evaluation is composed of a field test followed by a questionnaire. Field testing the application with volunteers was conducted using the following scenario: "*Two family members who are out together decide to split up because one of them wants to embark on an activity that the other is not very keen on undertaking, in this case, shopping at a fair. As a result, they schedule a time and a place to meet up so they can both go their separate ways and use their time as they see fit.*"

The tests were carried in three distinct locations, University of Minho's Campus, Mire de Tibães and Maximinos. All of these locations constitute a viable setting for the occurrence of a synchronisation activity as defined in the scope of our work.

We recruited 6 volunteers, chosen amongst friends whom we felt would be able to deliver a straightforward opinion in their evaluation. They had varying degrees of expertise in interfacing with touch based devices and more specifically the Android OS, which undeniably reflects in their opinions on the usability portion of the test.

After having set the scenario and contextualizing the experiment for them, they were presented with the Android device so they would be able to evaluate the system and draw their own conclusions. Upon finishing the field test and assimilating the experience, users were presented with a questionnaire. The part of the questionnaire related to usability was conducted in the moulds of a publicly available online heuristics questionnaire [8], this was fused together with an additional section of questions pertaining to our other more fundamental evaluation objectives, questions

which we thought were relevant and capable of directing the users to providing us the feedback we wanted. Some of the heuristics questions were adapted to better suit the purposes of our evaluation. Each questionnaire was answered in approximately 10 minutes and all information resulting from them stored in digital format. The only audiovisual record created was a demonstration of the working prototype.

5.2 Platform and test prototype

The platform used for testing is an HTC mobile device, model name Legend. The device comes equipped with an integrated GPS module, a 3.2" screen with a resolution of 320x480 (HVGA), running Android 2.1 (Éclair) with HTC Sense UI and all the input is touch driven.

The test prototype application uses GPS and the Android LBS API to sense a user's presence in the area chosen for an activity. It provides a help menu so that the user may clarify any doubts regarding what each of the controls do and regarding the process of activating a presence alarm for an activity. All activity data is collected using the calendar component that's provided in the Android OS, which is called from within our own application. This data is then stored in the devices SQLite database.

The following images present screenshots of the test prototype used in evaluation. In Fig. 2a we can see a detailed view of an on-going activity and its status. Fig. 2b shows the user receiving an update, Fig. 2c shows the content of the update, indicating that Mario has arrived to the activity location.

Figure 2. (a) Detailed view of an activity. **(b)** Statusbar notification for an activity. **(c)** Notifications pane with detailed description of the notification.

5.3 Analysis

Overall, the results obtained during the evaluation suggest that the system was positively perceived by the volunteers. In this section, we describe some of the main findings.

5.3.1 Usability

Regarding usability, opinions amongst testers were varied, some felt that the interface was simple, intuitive and to the point, while others felt it needed some refinement and glare. We have perceived that some of the problems that the users have identified were clearly connected with their lack of experience with touch based devices and especially the Android OS's UI interaction model.

Another issued raised by users was directly connected with the use of the Android notification model. We used this notification framework to warn users when someone arrives at the location of the activity. Users expressed concerns over the possibility of missing notifications, due to the default notification sound being too short and also, because there was no vibration or flashing LED warnings.

Other issues that were pointed out by some users regard the approach used to fill out the activity location field with GPS coordinates and the activation of a location alarm. Those users felt there should be a better way to get and set the GPS coordinates of an activity. Initially we had made that process automatic and transparent to the user, which on the face of it might seem ideal, but it posed a serious limitation in the way that it was done because it forced the user to be physically present at the point of rendezvous when setting the alarm, which is not ideal. So we opted for another approach which was to have the users press the GPS Coordinates button, which copies them to the clipboard and then have users paste them in the activity location field. This decoupled coordinate setting and alarm activation, so the users can have more freedom when it comes to the process of setting the meeting point for an activity. Still, we consider none of these approaches to be ideal and the application needs further refinement in this aspect. At the beginning of an activity, when still at the would-be meeting point, the alarm could be immediately activated. In doing so, the system will notify other users of an arrival, which was not the intention.

5.3.2 Main learnings from the study

The overall results suggest that users see great potential in a synchronisation tool, such as this one, as a method for replacing, or in some cases complementing, the traditional forms of synchronisation (SMS/phone call). The data suggests users testing the system find that the feedback given to them by the system is adequate and can easily substitute the one obtained via the traditional methods referred, thus validating our goal of facilitating interpersonal synchronisation in the context of an activity by extending the calendar as a tool of coarse-grained location based synchronisation.

User feedback showed that users are able to accompany and realize what the state of an activity is, as well as the repercussions of their actions, towards others and the system. This tells us that users trust the system and the information it relays to them, which is critical towards application viability in this context.

In spite of these promising results, users still expressed concerns that phones with the necessary capabilities may not be adequately priced, while others consider that the need for internet connectivity may result in them spending more money than they would with an SMS or a phone call. Other users pointed out other potential issues like internet connectivity and GPS connectivity driving down the device's autonomy. This issue is in part addressed by the Android platform itself with its advanced power management features, but it will continue to be mitigated by hardware evolution and also continued software evolution as Android is constantly evolving.

6 Conclusions

One of the main objectives was to explore the concept of coarse-grained interpersonal synchronisation using a calendar as an underlying tool and extending the calendar's functionality to provide coarse-grained location based synchronization. This is a goal we believe to have hit with a good measure of success, since the results of our evaluation with users suggest that the application developed to explore this concept seems to hold great potential and value for users.

In our study we were able to identify some issues in the ecosystem that could impact adoption of such a system. Factors like the current market and economic status quo in what regards mobile devices, namely smartphone pricing, adoption rate and internet data plans are a source of concern for users and may affect system adoption. Another key issue pertaining to the ecosystem is related to the architecture of the system, specifically, the shared calendar component. There is a definite need for one, but not obvious solution. Google Calendar is a viable option given that it's widely used and seems like a good approach to the issue, but its API is still not very matured. Ideally, our system should easily integrate with multiple types of calendar system, as we only make a very simple use of their features.

Privacy is also an important issue in such systems but in our research we were not able to determine any critical issues and users presented no objections.

As with any other project there is always work to be done in the future. At this point in time, we can point out an obvious issue to be addressed, which is to implement the remainder of the system. Additionally, there is room for optimizing the way polling to the synchronisation server is done. Factoring in information we already possess about planned activities, like the starting time and the type of activity, one can adjust the frequency with which the application polls the server. This would result in a better usage of battery and data, and would also contribute to the user having more up-to-date data at relevant times. Further into the future, one could extend the application functionalities. For instance, one feature we can see as being useful in a tool like this is to have the application interface with Google Maps to give the person directions to the meeting point. Users pointed out other interesting features like the production of graphics with user attendance and assiduity and possibly the ability to share them with friends.

Overall, we feel our research project is of valuable use to someone wanting to explore the underlying concept of synchronisation and that such an exploration could be carried out using the foundations we have laid down with our work.

7 References

[1] K. Tollmar and J. Persson, "Understanding remote presence," *Proceedings of the second Nordic conference on Human-computer interaction - NordiCHI '02*, New York, New York, USA: ACM Press, 2002, p. 41.

[2] C. Schmandt and N. Marmasse, "User-centered location awareness," *Computer*, vol. 37, 2004, pp. 110-111.

[3] L.A. Suchman, *Plans and Situated Actions: The Problem of Human-Machine Communication (Learning in Doing: Social, Cognitive and Computational Perspectives)*, Cambridge University Press, 1987.

[4] Carnegie Mellon University Mobile Commerce Lab, "Locaccino," 2009, http://locaccino.org/.

[5] V. Gundotra, "Google Latitude," *The Official Google Blog*, 2009, http://googleblog.blogspot.com/2009/02/see-where-your-friends-are-with-google.html.

[6] A. Sellen, R. Eardley, S. Izadi, and R. Harper, "The whereabouts clock," *CHI '06 extended abstracts on Human factors in computing systems - CHI '06*, New York, New York, USA: ACM Press, 2006, p. 1307.

[7] A. Sellen, R. Harper, R. Eardley, S. Izadi, T. Regan, A.S. Taylor, and K.R. Wood, "HomeNote," *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work - CSCW '06*, New York, New York, USA: ACM Press, 2006, p. 383.

[8] S.U. School Of Cognitive And Computing Sciences, "Interactive Heuristic Evaluation Toolkit," *Human-Centred Computer Systems Masters*, 2001, http://www.id-book.com/catherb/.

Ad Hoc Routing Under Randomised Propagation Models *

João Matos and Hugo Miranda

University of Lisbon
Faculty of Sciences
LaSIGE

Abstract. The deployment of mobile ad hoc networks is difficult in a research environment and therefore the performance of protocols for these networks has been mostly evaluated on simulators. A simulator must replicate realistic conditions and one of the most difficult aspects is the radio signal propagation model. The literature shows that many performance evaluations were conducted using propagation models that are not realistic for the expected application scenarios. This paper shows that the non-determinism present in some radio propagation models induce randomness which may compromise the performance of many protocols. To demonstrate the problem, this paper compares and discusses the performance of some routing protocols under different propagation models.

1 Introduction

Mobile Ad Hoc Networks (MANETs) are wireless networks with no fixed infrastructure and therefore are composed exclusively by the devices of the participants. All management and communication operations are assured by the participating devices. These networks are particularly relevant in scenarios where the deployment in advance of an infra-structure is not possible or desirable. Nodes communicate using their wireless network interfaces, which have a limited transmission range unlikely to cover all the nodes in the network. Message delivery is achieved by having nodes located between a source and a destination to retransmit the messages. Routing protocols are responsible for discovering a sequence of intermediate nodes (a route) that connects two endpoints.

Radio propagation considerably influences the performance of wireless communication systems, including ad hoc routing. The transmission path between two nodes can be a direct and unobstructed line-of-sight or a complex and strongly obstructed one, due to the presence of all kind of obstacles. Experimenting with wireless networks is usually done in simulated environments, because *i*) it is common for the number of devices involved to be high, *ii*) devices are often expensive and therefore it is wise to assure feasibility before deployment.

* This work has been partially supported by FCT multiannual fund and by project PATI (PTDC/EIA-EIA/103751/2008) through POSI and FEDER.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 379–390

There are many kinds of wireless networks, environments and radio technologies and all these aspects influence the effective signal propagation in the ether. As a consequence many radio propagation models have been devised. Unfortunately, some popular propagation models for network simulators do not account with multi-path propagation effects caused mostly by surrounding obstacles. The randomness caused by these unpredictable irregularities is frequently present in numerous types of radio wave propagation, including those used in most popular wireless network technologies, like IEEE 802.11 (WiFi).

Many ad hoc routing protocols ([1,2,3] to name a few) were tested under propagation models like *two-ray-ground* [4] and *free space* [5]. These propagation models are not adequate for testing realistic ad hoc networks (for example using WiFi technology in a region with obstacles). Therefore, the expected performance of many routing protocols may not be observed when used in a real deployment. This paper aims to highlight this problem through simulations by comparing the performance of three routing protocols using two different radio propagation models. Results confirm our expectations by showing a significant performance degradation in a more realistic propagation model. The paper also dissects these results and identifies the design characteristics of the protocols that make them more vulnerable. The paper is organised as follows: in Sec. 2 the routing protocols in comparison are presented. Section 3 describes the most relevant propagation models and Sec. 4 addresses the adaptation problems of routing protocols to specific radio propagation models. The evaluation results are discussed in Sec. 5 and in Sec. 6 the related work is presented.

2 Routing Protocols

Depending on their eagerness in populating routing tables, routing protocols for MANETs can be arranged in two broad categories: reactive and proactive. Reactive routing protocols are distinguished by having routes being discovered on-demand, while proactive routing protocols aim to keep their routing tables permanently up-to-date. For completeness, our study focused on protocols of both categories. The following presentation is oriented to the aspects relevant for our evaluation. The interested reader is referred to [1,2,3,6] for in-depth descriptions of these protocols.

2.1 Reactive Routing Protocols

In reactive routing protocols routing tables are filled and updated during route discovery operations, which are initiated only when a route to a certain destination is required and is absent on the routing table. The node requiring a route for an unknown destination broadcasts a *route request* message, disseminated to the entire network.The most simple and popular way to deliver a message to the entire network is to flood it, that is, to have all the nodes retransmitting it when it is received for the first time. This broadcast algorithm is called flooding and is used by many reactive routing protocols for route discovery operations.

When a node receives a *route request* message for the first time, it verifies if its routing table contains a route to the required destination and if not, continues the propagation of the *route request*. Otherwise, the node sends a point-to-point *route reply* message addressed to the source of the *route request*. The *route reply* message will follow the route created during the propagation of the *route request*. That is, each node broadcasting the *route request* message must keep track of the node from which it was received. The destination node also produces a *route reply* when a *route request* message is received.

Node's movement, network congestion and multi-path propagation effects frequently invalidate routes. *Route Error* messages are notifications addressed to the source of some data message and produced by intermediate nodes unable to deliver the message to the next hop.

In our study, the performance of reactive routing protocols was evaluated using two of the most representative routing protocols of this class, which are detailed below.

AODV The route request message of the Ad hoc On-demand Distance Vector (AODV) routing protocol [3] includes, among others, fields for the sender's address, destination's address and broadcast id. The pair <sender's address, broadcast id> of each *route request* message allow nodes to detect duplicates. Other fields, like *sender sequence number* and *destination sequence number*, allows nodes to determine the freshness of the route. The sequence number for each destination is stored in the routing table, together with the number of hops to the destination and the address of the *next hop*, that is, the node to whom messages addressed to the destination should be relayed.

Routes are learnt in the opposite direction of message propagation. That is, the reception of a route request or route reply message is used by nodes to learn a route to the sender of the message. The *next hop* for this route will be the node from which the message was received.

AODV purges from the routing table routes that have not been used for a predefined time. In addition, it updates its routing table if: *i*) the sequence number of the new route is strictly higher or; *ii*) the sequence number is equal but the number of hops to the destination is lower. One aspect of AODV very relevant for this paper is that every node replies only once to the same route request. This means that if a node receives the same route request from several neighbours, it replies only to the neighbour who first delivered the route request.

DSR The structure of the routing table in the Dynamic Source Routing (DSR) protocol [2,6] is significantly more complex as it stores complete routes. In addition, nodes cache multiple routes to any destination. This allows a faster reaction to routing changes given that there will be no additional overhead from a new route request operation. DSR data packets carry the full list of nodes that should be traversed to reach the destination.

During the propagation of a route request, each intermediate node appends its address to the header of the route request message, thus providing the complete sequence of intermediate nodes that lead to the destination.

One important aspect of DSR for our work, is that whenever a *route error* message is received by the source, it tries all the routes present in its cache, before starting a new route discovery operation. When a new *route request* message is disseminated, it carries information about the broken links found in the routes cached by the source. The network interfaces of nodes running DSR are expected to operate in promiscuous mode, receiving and interpreting every message sent by any neighbour. Listened messages are used to update node's routing table. Examples of applications of promiscuous mode are the learning of new routes listened from data packets and the removal of stalled routes learnt from snooping route error messages.

2.2 Proactive Routing Protocols

In proactive routing protocols every node maintains in its routing table an up to date list of all participants and routes to reach them. This is achieved by having nodes to periodically broadcast their routing tables.

Each node in the network maintains, for each destination, a preferred neighbour and each data packet contains a destination node identifier in its header. When a node receives a data packet, it forwards the packet to the preferred neighbour for its destination. The methods used to construct, maintain and update routing tables differ between various routing protocols.

The proactive routing protocol Destination-Sequenced Distance Vector (DSDV) [1] requires each mobile node to advertise its own routing table to its 1-hop neighbours. Routing tables include all available destinations with the respective routes and the number of hops. The entries in this list may change dynamically over time, so the advertisement must be made often enough to avoid unavailability problems. When significantly new update information is available, nodes transmit it immediately.

In a very large population of nodes, adjustments are likely to be made a short while after an exchange of complete routing tables. In order to reduce the amount of information exchanged, two types of packets are defined. One carries all the available information, and is called *full dump* and the other possesses only the information changed ever since the last *full dump*, called *incremental*.

3 Propagation Models

Propagation models are used in simulators to predict the received signal strength indicator of each packet received by a node. Propagation models that predict the mean signal strength for an arbitrary distance between two nodes are called *large scale* propagation models, because these distances may become very large. This section covers this type of propagation models and presents three common methods for received signal strength prediction.

The *path loss*, which represents signal attenuation as a positive quantity measured in dB, is defined as the difference between the transmitted power and the received power. Different propagation models may be distinguished by

the method used to calculate the path loss between two nodes. Therefore, the received signal strength is predicted by the subtraction between the effective transmitted power and the path loss calculated.

The popular network simulator *ns-2*[1] in particular, creates a threshold variable which defines the minimum possible value of the Received Signal Strength Indicator (RSSI) with which a node is still able to receive a packet. Considering the propagation model in use, it then calculates the RSSI with which a packet was received by a node. If the value is smaller than the threshold, *ns-2* considers that the packet was not received by the node. The following sections present three popular distinct propagation models available in *ns-2*.

3.1 Deterministic Models

The free space propagation model [5] is a deterministic propagation model that defines the communication range as a perfect sphere around the transmitter. In free space only one clear and unobstructed line-of-sight path between the transmitter and receiver exists. The received signal strength indicator is calculated by the Friis free space equation $Pr(d) = \frac{PtGtGr\lambda^2}{(4\pi)^2 d^2 L}$, where d is the distance between nodes, P_t is the transmitted power signal, G_t and G_r are the antenna gains of the transmitter and the receiver respectively, L is the system loss and λ is the wavelength in meters. The free space propagation model is considered accurate to predict *rssi* for satellite communication systems and microwave line-of-sight radio links [4].

In a mobile radio channel, a single direct path between the base station and a mobile node is seldom the only physical means for propagation, and hence free space is in most cases inaccurate when used alone [4]. Instead of having a single line-of-sight path between two nodes, the two-ray ground reflection model considers both the direct path and a ground reflection path, as shown in Fig. 1. The total received electrical field (E_{TOT}) is the result of the direct line-of-sight component (E_{LOS}) and the ground reflected component (E_g). This model gives more accurate prediction at a long distance than the free space model [4]. However, the two-ray ground reflection model is also deterministic when predicting the received signal strength indicator. It is calculated using the formula $P_r(d) = \frac{PtGtGrht^2hr^2}{d^4 L}$, where h_t and h_r are the heights of the transmit and receive antennas respectively. Like in free space, the communication range in the two-ray ground reflection model is an ideal circle, centred at the transmitter. This model has been considered reasonably accurate for mobile radio systems that use tall towers and also for line-of-sight microcell channels in urban environments [4].

3.2 Randomized Models

The models above do not consider the fact that the surrounding clutter may be very different at two different locations having the same distance to the source

[1] http://www.isi.edu/nsnam/ns/

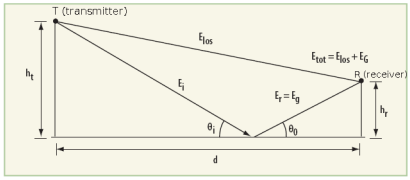


Fig. 1. Two-ray Ground Reflection Model [4]

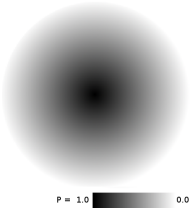


Fig. 2. Log-normal Shadowing: (P is the probability of receiving a packet)

or even for the same location at different moments in time. Therefore their use is inappropriate, in various scenarios because the received power is actually affected by unpredictable multi-path propagation effects. The Log-normal shadowing model considers that the signal fades log-normally and randomly. That is, the path loss increases log-normally with distance but a random component, whose influence becomes more visible as the path loss increases, must also be considered. In practice, the model results in having nodes located farther from the transmitter possibly receiving packets while some nodes located closer might not. This also means that the probability of a node receiving a message becomes smaller as the distance increases, as illustrated in Fig. 2.

Like most propagation models, the shadowing propagation model determines the received power at distance d removing the calculated path loss value from the transmitted power value, as shown in Eq. 1. However, as shown in Eq. 2, the path loss is divided in two parts. One part is the log-distance path loss model and predicts log-normally the mean received power at distance d, denoted by $\overline{PL}(d)$ (Eq. 3). This part uses a close-in distance d_0 as a reference. The second part of the model consists on the variation of the received power at a certain distance. It is a zero-mean Gaussian distributed random variable (in dB) with standard deviation σ (also in dB). Therefore, considering Eq. 2, the variable $X\sigma$ represents the random part of the model and the variable $\overline{PL}(d)$, the deterministic part.

$$P_r(d)[dBm] = \underbrace{P_t[dBm]}_{\text{Transmitted power}} - \underbrace{PL(d)[dB]}_{\text{Path loss}} \tag{1}$$

$$PL(d)[dB] = \underbrace{\overline{PL}(d)}_{\text{log-normal path loss}} + \underbrace{X\sigma}_{\text{Random path loss}} \tag{2}$$

$$\overline{PL}(d) = \overline{PL}(d_0) - 10\beta\ log(\frac{d}{d0}) \tag{3}$$

The log-normal distribution describes the random shadowing effects which occur over a large number of measurement locations which have the same distance to the source, but have different levels of clutter on the propagation path [4]. The close-in reference distance d_0, the path exponent β and the standard deviation σ, statistically describe the model for an arbitrary location. It

should be noted that, in contrast with two-ray ground and free space, log-normal shadowing does not assume the communication range to be a perfect sphere.

4 Routing Protocols and Propagation Models

Energetic, communicational and computational resources of the devices in a MANET are usually limited. Networking operations, namely transmissions are expensive in terms of energy consumption [7] and routing protocols aim to reduce them to a minimum. Metrics used by routing protocols usually combine cost and congestion, evaluated respectively by the number of hops and delay. The randomness imposed by fading effects is usually neglected and the protocols tend to adapt poorly to shadowed environments. In this paper, we identify two adaptation problems that can be observed in some of the most popular routing protocols for MANETs.

Shadowing induced link asymmetries [8,9] In the shadowing model, neighbour nodes farther from the source have a low probability to receive the route discovery message than closer ones (as illustrated in Fig. 2). However if a node has many neighbours, chances are that at least one of the distant neighbors does receive it. Additionally, the route discovery message usually travels through multiple hops and therefore, it is likely for a route to include at least one such "weak" (long) link.

Surprisingly, routing protocol metrics (like those used by AODV, DSR and DSDV) tend to favour routes that include weak links as they are expected to have a lower number of hops (thus reducing cost) and to be discovered faster (which is interpreted as a sign of lower congestion). Although this route would indeed be preferable, the weak link has the same low probability of delivering the route reply in the reverse path. Probabilities suggest that in most cases, the route ends up not being established. When the waiting period expires, the source will be required to start a new route discovery operation which might be unsuccessful for the same reason.

Despite not having *route request* nor *route reply* messages, proactive routing protocols are also affected by this problem. Nodes exchange routing information through periodic messages which are likely to contain routes including some weak link. Again, these routes are likely to be preferred because metrics suggest they have a better performance.

Route stability in a shadowed environment The second problem appears after a route between two nodes has been established. In deterministic models like *free-space* and *two-ray ground* if a node is located within the transmission range of another, it will certainly receive every packet sent. In practice, a route does not break unless some node that composes it moved away or a congestion problem made some node believe that its neighbour moved. On the other hand, a propagation model such as *shadowing* does not guarantee that a node close enough to the sender will receive the message. Such transient problems are usually handled

at the link layer level, at the expenses of additional traffic produced by retransmissions. However, in some cases, the time took by the link layer to deliver the packet can be misinterpreted by the routing protocol as a sign of route breakage. More frequent route invalidation result in additional traffic produced by route errors and route discoveries.

5 Evaluation

To validate the two hypothesis stated in Sec. 4, we analyse the performance of three routing protocols, AODV, DSR and DSDV under two radio propagation models, *two-ray ground* and *shadowing*. The goal is to look for patterns that appear in the performance of the routing protocols while using *shadowing* propagation model and are not present when *two-ray ground* is used. Both *free space* and *two-ray ground* are deterministic propagation models and represent the transmission range as an ideal sphere and therefore including the two models in our evaluation would not provide any additional contribution. Three routing protocols, two reactive and one proactive are used to evaluate if the problems are exclusive to one particular protocol or class of routing protocols. Results are obtained using v. 2.34 of the *ns-2* network simulator. This simulator already implements all the propagation models and routing protocols experimented.

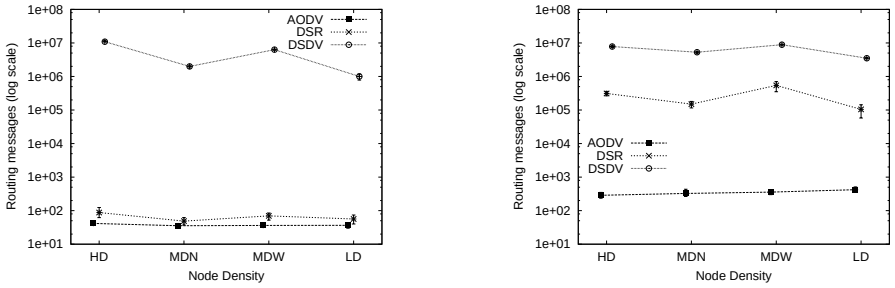
Simulation Test Bed The performance of routing protocols is affected by a myriad of factors like mobility and congestion. The experiments presented in this paper aimed to reduce to a minimum the interference on performance of external factors not strictly related with the propagation model. Therefore, we defined a baseline scenario of quasi ideal conditions for any of the protocols. To avoid congestion, traffic is kept constant at a low rate of one 512 bytes data packet per second. To enforce route discovery operations the source and destination of the packets changes every 60 seconds. Nodes do not move for the entire extent of the simulations, thus preventing "legitimate" route errors and additional route discovery operations.

Experiments consist of 160 simulations for each pair of routing protocol and propagation model. Each simulation has the duration of 1800 seconds. To evaluate the impact of the number of neighbours and route length on the performance of each pair, the simulations have been arranged in 4 different scenarios, presented in Tbl. 1. In each simulation nodes are randomly deployed over a region with the specified dimension according to an uniform distribution. An uniform distribution is also followed on each simulation to define the traffic sources and destinations. To make comparisons acceptable, the exact same conditions of node deployment and traffic are used for every pair of routing protocol and propagation model. Plots present the average of the 40 simulations for each <routing protocol,propagation model,scenario> tuple. Error bars depict the values observed for the 10% lowest and highest simulation.

In the two-ray ground propagation model each node was configured for a transmission range of $250m$. The shadowing simulations test an outdoor shadowed urban area with a path loss exponent β of 2.7 and a standard deviation σ

Name	Stands for	Density	Nodes	Region Size
HD	High Density	$3750m^2.node$	200	$1500m \times 500m$
MDN	Medium Density Narrow area	$7500m^2.node$	100	$1500m \times 500m$
MDW	Medium Density Wide area	$7500m^2.node$	200	$3000m \times 500m$
LD	Low Density	$15000m^2.node$	100	$3000m \times 500m$

Table 1. Comparison of the configurations experimented



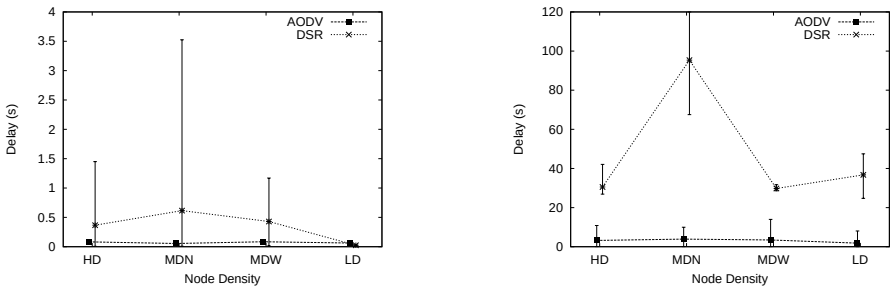
(a) Two-ray ground (b) Shadowing

Fig. 3. Route discovery messages

of 4, with a 95% of correct reception at $250m$ [10]. These are values commonly used in ad hoc routing experiments ([8,11] for example). We recall that an exact range cannot be defined for the shadowing propagation model. Therefore, with some probability, some nodes closer than $250m$ from the transmitter do not deliver a packet while others, more distant will.

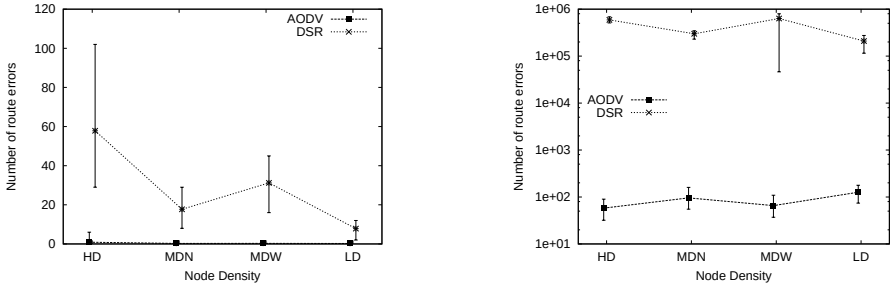
Evaluation Results The number of route request messages originated by each tested protocol are depicted in Fig. 3. For DSDV, the plots depict the number of periodic route advertisement messages that is characteristic of proactive routing protocols. The figures show that in all three protocols, considerably more routing messages are originated when the shadowing propagation model is used. This is more significant on DSR that suffers an increase of more than 1000 times.

Confirmed the negative impact of the shadowing propagation model on the number of route discovery operations, we proceed to investigate the origin of the problem. Figure 4, that depicts the average time between the triggering of a route discovery operation and the reception of the first route reply, confirms the presence of the *Shadowing induced link asymmetries* problem. Knowing that no obstacle is made to the message propagation speed by any of the propagation models, an increased delay in the delivery of the route replies can only be attributed to the need of the route discovery initiator to perform multiple retries. Again, the problem is more visible in DSR, what follows naturally from the observed increase in the number of route requests that have been initiated. We note that for DSDV the delay is always null because routes are immediately available on the route cache of any node.



(a) Two-ray Ground (b) Shadowing

Fig. 4. Route discovery latency



(a) Two-ray Ground (b) Shadowing

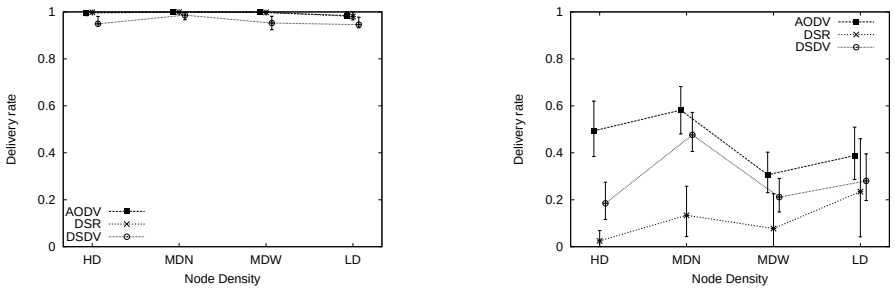
Fig. 5. Number of route errors

However, *Shadowing induced link asymmetries* is not the only problem affecting DSR and AODV. Figure 5, counts the number of observed route error messages and confirms that in the shadowing propagation model routes: *i*) are equally established and *ii*) break far more often than in deterministic propagation models. Because nodes do not move during the simulation, *ii*) supports the conclusion that the *Route stability in a shadowed environment* problem is equally present. Again DSDV is not accounted for this metric because route invalidation is detected within the periodic exchange of routing information.

The consequences of the problems discussed above are depicted in Fig. 6. In Fig. 6(a) all protocols present an average delivery over 95% for all topologies. On the other hand, the delivery ratio in Fig. 6(b) is bellow 60% for AODV, 50% for DSDV and 30% for DSR.

6 Related Work

In [8] a comparison of these three routing protocols for Wireless Sensor Networks under *two-ray ground* and *shadowing* propagation models was also preformed.



(a) Two-ray Ground (b) Shadowing

Fig. 6. Delivery rate

However, the effects induced by the shadowing propagation model described above are not identified nor described in the paper and the analysis presented does not provide the same conclusions as this paper. The authors focus mostly on the properties of the wireless sensor network studied and give little attention to the radio propagation models and to their relevance in routing. In addition, the delivery rate is the only metric presented and therefore does not support the conclusion that the shadowing problems described above are actually present. The authors continued their work and presented a similar study for Mobile Event using AODV [11].

Solutions for these effects are very few. Studies about the minimum node density required to achieve a connected large-scale ad hoc network, where every node has the same transmitting and receiving capabilities under a shadowed environment are presented in [12,9]. The authors in [13] created a sub-layer between the network and the MAC layer that provides a bidirectional abstraction of a shadowed environment for routing protocols. Another example of attenuation for these problems was presented on [14] where the authors propose a model for estimation of the bit error rate for each link made available to a node. The use of MIMO devices and multiple frequency networks may diminish considerably the problems discussed in this paper. However, transmission errors are an integral part of the wireless propagation medium and are not expected to be fully avoided.

7 Conclusions and Future Work

Mobile ad hoc networks (MANETs) are a promising technology for a number of scenarios. However, they present a networking environment that is considerably different of what can be found in wired networks. An effective deployment of MANETs is not possible without a realistic estimation of the performance of a number of protocols that are fundamental for MANETs expected applications. This paper compared the performance of 3 routing protocols when distinct signal propagation models are simulated. The paper shows that all protocols have

a significant performance degradation when the log-normal shadowing propagation model is used. Unfortunately, this is the most realistic model for expected MANET deployment scenarios.

As future work, authors plan to extend this study to other protocols and to devise mechanisms that may help to attenuate the difficulties observed by these protocols to cope with the shadowing propagation model. The apparently better resilience of AODV to transient connectivity in comparison with DSR will be used as an important guideline in our future work.

References

1. Perkins, C.E., Bhagwat, P.: Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In: SIGCOMM '94: Proc. of the Conf. on Communications architectures, protocols and applications. (1994) 234–244
2. Johnson, D.B., Maltz, D.A., Broch, J.: DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. In: Ad Hoc Networking. Addison-Wesley (2001) 139–172
3. Perkins, C.E., Royer, E.M.: Ad-hoc on-demand distance vector routing. In: Procs. of the 2nd IEEE Works. on Mobile Comp. Systems and Applications. (1999) 90–100
4. Rappaport, T.: Wireless Communications: Principles and Practice. Prentice Hall PTR (2001)
5. Friis, H.T.: A note on a simple transmission formula. Proc. IRE **34** (1946)
6. Johnson, D.B., Maltz, D.A.: Dynamic Source Routing in Ad Hoc Wireless Networks. In: Mobile Computing. Kluwer Academic Publishers (1996) 153–181
7. Feeney, L.M., Nilsson, M.: Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In: IEEE Infocom. (2001) 1548–1557
8. Yang, T., Ikeda, M., De Marco, G., Barolli, L.: Performance behavior of AODV, DSR and DSDV protocols for different radio models in ad-hoc sensor networks. In: Parallel Processing Workshops, 2007. ICPPW 2007. Int'l Conf. on. (2007)
9. De Marco, G., Longo, M., Postiglione, F.: Connectivity of ad hoc networks with link asymmetries induced by shadowing. Communications Letters, IEEE **11**(6) (2007) 495 –497
10. Greis, M.: The ns manual (2010)
11. Yang, T., Ikeda, M., Barolli, L., Durresi, A., Xhafa, F.: Performance evaluation of wireless sensor networks for different radio models considering mobile event. In: Complex, Intelligent and Software Intensive Systems (CISIS), 2010 Int'l Conf. on. (2010) 180–187
12. Bettstetter, C., Hartmann, C.: Connectivity of wireless multihop networks in a shadow fading environment. Wirel. Netw. **11**(5) (2005) 571–579
13. Ramasubramanian, V., Chandra, R., Mosse, D.: Providing a bidirectional abstraction for unidirectional ad hoc networks. In: INFOCOM 2002. 21st Joint Conf. of the IEEE Computer and Communications Societies. Proc. IEEE. Volume 3. (2002) 1258–1267
14. Agrawal, R.: Performance of routing strategy (bit error based) in fading environments for mobile adhoc networks. In: Personal Wireless Communications (ICPWC). IEEE Int'l Conf. on. (2005) 550–554

Decentralized Processing of Participatory Sensing Data*

Heitor Ferreira, Sérgio Duarte and Nuno Preguiça

CITI / Dep. de Informática - Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa,
Quinta da Torre, 2829 -516 Caparica, Portugal

Abstract. Participatory Sensing is an emerging application paradigm that leverages the growing ubiquity of sensor-capable smartphones to allow communities to carry out wide-area sensing tasks, as a side-effect of people's everyday lives and movements. This paper proposes a decentralized infrastructure for supporting Participatory Sensing applications. It describes an architecture for modeling, prototyping and developing the distributed processing of participatory sensing data. Our initial evaluation shows that the proposed architecture can efficiently distribute the load among participating nodes.

Keywords: participatory sensing, decentralized processing, data streaming, mobile computing

Sumário. Participatory sensing é um novo paradigma aplicacional potenciado pela crescente difusão de telemóveis equipados com sensores, permitindo a colecção de dados sensoriais em áreas alargadas, por uma comunidade de utilizadores, aproveitando as actividades e percursos do seu quotidiano. Este artigo propõe uma infra-estrutura distribuída de suporte a aplicações Participatory Sensing. É descrita uma arquitectura para a modulação, prototipagem e desenvolvimento de processamento distribuído de dados de participatory sensing, tendo como objectivo o desenvolvimento mais fácil e rápido deste tipo de aplicações. A avaliação inicial mostra que a arquitectura proposta permite distribuir a carga eficientemente entre os nós participantes no sistema.

Palavras-chave: Sensoriamento participativo, processamento distribuído, data streaming, computação móvel

1 Introduction

Participatory Sensing [2,3] is a new application paradigm that aims to turn personal mobile devices into advanced mobile sensing networks. Thanks to the

* This work was partially supported by FCT/MCTES, project #PTDC/EIA/76114/2006 and CITI.

willingness of the users and the inherent mobility of their daily routines, it is possible to assemble detailed views and interpretations of the physical world without the costs associated with the deployment of dense, wide-area, sensing infrastructures. Examples of the potential of this paradigm already exist, such as experiments combining accelerometer and GPS data to monitor road conservation [5] and traffic congestion [8,11]. The expected outcome from this new movement is the creation of rich data sets, a data commons, supporting new services, discourses and interpretations.

The first wave of case-study applications already shows the potential of this paradigm, but their relatively confined specificity also exposes their exploratory stage. In general, these applications stand on top of adapted middleware architectures, in many cases using centralized entities [5,8,11], or limited distribution models. The more encompassing efforts at platform support tend to originate from fixed sensor networks backgrounds, with limited allowances for mobility [10,7]. Centralized solutions raise several problems. On one hand, there are the implications of having a centralized repository hosting privacy sensitive information. On the other hand, a centralized model has financial costs that can discourage community-driven initiatives.

This paper focuses on the issue above and proposes a decentralized infrastructure for supporting participatory sensing applications, whose goal is to ease the prototyping and development of participatory sensing applications. The proposed system includes a framework for modeling and carrying out the processing of participatory sensing data in a decentralized, fully distributed fashion. Our initial evaluation shows that the proposed architecture can efficiently distribute the load among nodes, thus tackling the issues mentioned above.

The rest of the paper is organized as follows. Section 2 describes the overall system model, including the proposed system architecture, programming abstractions and execution environment. A case-study application and experimental results are presented and discussed in Section 3. Section 4 presents a review of related work and Section 5 concludes the paper with our plans for future work and some final remarks.

2 System model

2.1 Architecture

The proposed distributed system for running participatory sensing applications consists of a high number of mobile nodes equipped with sensors, and a fixed support infrastructure. A mobile node can be any mobile computing platform, typically a mobile phone, and is expected to have limited resources, in particular in terms of computing power and battery life. Mobile nodes are connected to a fixed infrastructure, composed by a set of nodes structured in an overlay network, that supports the more resource intensive operations, such as data processing, routing and storage. Fixed nodes can be personal computers, virtual machines running in a utility computing infrastructure, or servers hosted by independent entities.

Applications are hosted in this hybrid environment, supported by a service middleware running in both mobile and fixed nodes. On the fixed infrastructure, an *application context* defines data acquisition, processing and storage requirements, while on the mobile side, contexts support the data acquisition tasks. Applications clients - hosted on mobile nodes or, in case of desktop or server based clients, connected directly to the fixed infrastructure - issue queries and receive result streams through the service middleware, thus supporting the interaction between user and service.

2.2 Data Processing Model

Streams Participatory sensing applications manage a continuous flow of data resulting from sensing and data processing activities. Generally, a stream is a sequence of data tuples with a common structure and semantic, represented as a set of named attributes. Regarding its origin, a *data acquisition stream* is a tuple sequence produced by mobile nodes according to the acquisition requirements expressed by an application. This *raw* data has a spatial and temporal nature - a timestamp records the time when the sensor reading was sampled, while physical coordinates, or a spatial extent, convey the location in space that the sample refers to. A *derived steam* results from the application of a transformation operation over a source stream.

Virtual Tables Similarly to the relational paradigm, where a data set with a common schema is represented by a relation, or table, *participatory sensing* applications model data by defining *virtual tables*. A virtual table specifies a derived stream in terms of one or more inputs - either a set of data acquisition streams or another virtual table - and a sequence of stream operators structured in a processing *pipeline*. The term virtual is used here because tables do not necessarily have associated storage - although the same abstraction can be extended to support persistence by storing and replaying a previous *live* stream. A *query* expresses a spatial constraint over a virtual table - a bounding box, for instance. The result of a query is a continuous tuple stream produced by the target virtual table, resulting from the application of the spatial restriction over its base stream.

Stream Transformations A stream transformation can be modeled as a sequence of *stream operators* structured in a processing *pipeline*. Data tuples flow in a pipeline, being processed in sequence by each operator - transforming, aggregating, filtering and classifying data. Following is a description of the stream operators considered in the context of this work.

Processing and Filtering The *processor* operator is the main extension basis for the implementation of domain specific processing, such as interpolation of sensor readings, unit conversion and data *mapping*. Mapping classifies data according to an uniform representation - e.g., a grid over the geographical space or the buckets in an histogram - thus establishing relations between data in order to support aggregation operations. Spatial mapping operations assign a spatial extent to data tuples, such as a bounding box, or physical coordinates representing the centroid of the extent.

Partitioning The *groupBy* operator partitions data by specific tuple attributes, or an arbitrary partitioning condition, producing independent data streams. Each independent stream is processed by a sub-pipeline specified by the operator - for simplicity, each sub-pipeline cannot include itself a *groupBy* operator. Partitioning is used to group related data, for instance by creating independent streams for data in particular cells, according to an uniform grid introduced by a mapping operator.

Stream decomposition To aggregate data, continuous streams have to be broken down into discrete tuple sequences. A *timeWindow* partitions the stream into possibly overlapping time periods using a sliding window.

Aggregation An *aggregator* operates over a finite tuple sequence applying operations, such as maximum, minimum, count, sum and average, over one or more input attributes of the input tuples. An aggregator can be used together with mapping, partitioning and stream decomposition in order to continuously produce independent aggregate values over the spatial decomposition defined by the mapping operator.

Classifier A *classifier* is a specialized processor used to generate inferences from aggregated data, such as detecting an event. A classification tuple is forwarded whenever an input aggregate is *complete* and satisfies an application defined condition. An aggregate tuple is complete when it takes into account all the information bounded by its spatial extent - see section 2.3.

2.3 Distribution Strategy

A stream transformation pipeline can be broken down into two stages, or roles - *data sourcing* and *global aggregation*. Data sourcing refers to the process through which each node produces partial state tuples from the continuous sensor input received from mobile devices, while global aggregation refers to the production of an aggregate result by merging the partial states from several nodes. One key aspect of the proposed system is that processing of these stream transformation operations is performed in a fully distributed way, using a strategy called QTree, as explained next.

In QTree, each fixed node is assigned a latitude and longitude. Data partitioning is based on subdivision (or *splitting*) of geographic space into quadrants that hold at least a minimum number of nodes (the *minimum occupancy*). Each node belongs simultaneously to all the quadrants that contain it, down to the smallest - called its *maximum division quadrant*. Mobile nodes upload sensor readings to an acquisition node whose physical coordinates lie in the same maximum division quadrant (*M1* and *M2* in figure 1). Areas with low node density can result in data dispersion across the entire node base; to reduce query scope, QTree assumes a *minimum division level* - meaning that geographic space is fully divided at this level i.e., all partitions have minimum occupancy.

Query Distribution To reach all relevant data, a query has to be distributed to all nodes within its *search area*, defined as the union of quadrants, at the minimum division level, that completely cover the query area. This is illustrated

in figure 1, where the shadowed quadrant represents the search area for query Q.

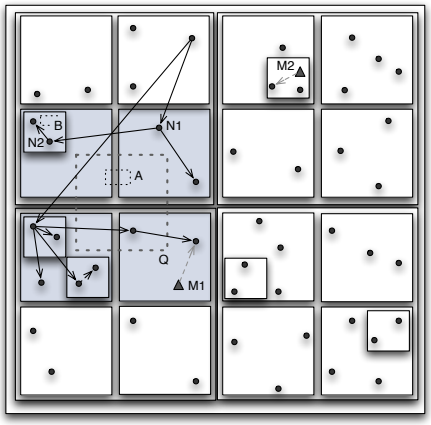


Fig. 1: QTree spatial partitioning with a minimum occupancy of 2 nodes and minimum division level of 2

A query is disseminated by recursively subdividing the search area until all nodes are reached, building a distribution tree in the process as depicted in figure 1. Initially, the root node divides the world into four quadrants, finds their interceptions with the search area and forwards the query to a randomly selected peer in each interception. Target nodes repeat the same procedure; for quadrants that do not have minimum occupancy, the node assumes the aggregation role and forwards the query to all peers in the area - these nodes become the tree leaves, acting as data sources. QTree provides an inherent balancing mechanism, given that a different node is chosen each time the aggregation tree is built and, the wider the aggregation area, the larger the set of candidate nodes.

Query Processing Aggregation is performed using the reverse path of the query dissemination tree. An important aspect of QTree, that allows the reduction of computation and communication costs, is that aggregate tuples are *complete* at the tree level where the assigned quadrant completely encloses its spatial extent - i.e., upper tree levels will not hold additional data regarding that extent - and can be forwarded to the query root without further processing. In figure 1, extents *A* and *B* are complete at the nodes *N1* and *N2* respectively. Another relevant characteristic in QTree is that although nodes closer to the root cover wider areas, they only aggregate data for spatial extents that are not completely enclosed at lower levels of the aggregation tree.

Network Dynamism Node failures in an aggregation tree impact query results, the severity of this impact depending on how close to the root the failure occurs. Failure requires rebuilding the tree for the affected quadrant, possibly after merging imposed by the minimum node occupancy requirement. When a node joins the network during query execution it can be incorporated into the aggregation tree after any necessary subdivision of space and consequent tree restructuring.

3 Evaluation

The system presented in the previous sections has been evaluated with two goals in mind. Firstly, to obtain a initial assessment of the expressivity of the proposed programming abstractions and, secondly, to evaluate in quantitative terms the performance of the distribution strategy that has been adopted. To this end, we modeled and implemented a case-study application, in a simulation setting, focused towards realtime participatory sensing data processing. This application, called SpeedSense, continuously monitors the current traffic status in an urban setting to allow client applications to access the current traffic speed per road and information about congested roads. For this purpose, simulated users collect real-time data while driving, using GPS equipped mobile devices. While, at this point, this case-study does not intend to be a realistic implementation of road traffic estimation, the scenario involved is a paradigmatic one for Participatory Sensing and is featured in some of the most referenced works in the area [8,11].

3.1 Case study - SpeedSense

SpeedSense infers the current average speed, and congestion detections in a given area, based on GPS data sampled by in-transit vehicles at periodic intervals. For this purpose, two virtual tables have been designed: TrafficSpeed, which supports querying for average speed per road segment, while the other, TraficHotspots, allows for querying for congestion detections.

For the road network (and traffic) model, SpeedSense requires a map representation of the application's geographic area of coverage. The Open Street Map (OSM) [12] vectorial representation of the road network is used for that purpose. This map data is used to map geographic coordinates to road segments, to determine the spatial extent of segments and their associated road type. The network model used in the evaluation divides roads, as needed, into segments with a maximum of 1 km and uses separate segments for each driving direction. Each road is assigned an expected (uncongested) driving speed according to its type: highway, primary to tertiary and residential.

TrafficSpeed Virtual Table This table derives directly from the *GPSReading* sensors input and produces an output stream of *AggregateSpeed* tuples that convey a segment, sample count, and total and average speed. Average speed is computed using a time-window to break down the continuous acquisition stream into finite time intervals (cf. Definition 1). Specifically, the data sourcing pipeline stage handles local aggregation of raw GPS input data, by mapping incoming samples to road segment using the *process* operator (for simplification, raw GPS readings reference the segment identifier); a *timeWindow* accumulates data samples for the given time period, then *groupBy* partitions the resulting data into independent substreams for each segment. For each of these, *aggregate* accumulates data samples for the given time period to compute the intermediate sum and count results that are used to produce the actual (moving) average speed for that segment as an *AggregateSpeed* tuple.

The global aggregation stage receives *AggregateSpeed* values from descendent peers and produces the overall average speeds by merging the partial records.

A *timeWindow* and operator *groupBy* are again used to accumulate data and partition the stream. For each of the resulting partitions (or substreams) *aggregate* is once more used for summing and counting all partial contributions and produce the actual *AggregateSpeed* value at this peer.

Definition 1 TrafficSpeed virtual table specification

```
sensorInput( GPSReading )
dataSource {
   process{ GPSReading r ->
      r.derive( MappedSpeed, [boundingBox: model.getSegmentExtent(r.segmentId) ])
   }
   timeWindow(mode: periodic, size:15, slide:10)
   groupBy(['segmentId']){
      aggregate( AggregateSpeed ) { MappedSpeed m ->
         sum(m, 'speed', 'sumSpeed')
         count(m, 'count')
} } }
globalAggregation {
   timeWindow(mode: periodic, size:10, slide:10)
   groupBy(['segmentId']){
      aggregate( AggregateSpeed ) { AggregateSpeed a ->
         avg(a, 'sumSpeed', 'count', 'avgSpeed')
} } }
```

TrafficHotspots Virtual Table This table outputs a stream of *Hotspot* tuples representing real-time detections of congested road segments. It is based on a simple model that compares the current average speed of a segment against a *congestion threshold*, given as a fraction of the maximum speed for that particular road, obtained from the road network model. It also takes into account the number of samples used to compute the average speed of the segment to provide a measure of the confidence or reliability in a detection result. Refer to Definition 2 for the actual specification of this table, where it can be seen that it derives from the TrafficSpeed table and, essentially, extends its global aggregation stage with the hotspot detection classifier. This *classifier* operator receives an *AggregateSpeeds* stream and produces a *Hotspot* tuple whenever the computed average is complete, reliable and below the congestion threshold.

Definition 2 TrafficHotspots virtual table specification

```
tableInput("TrafficSpeed")
globalAggregation {
   classify( AggregateSpeed ) { AggregateSpeed a ->
      if(a.count > COUNT_THRESHOLD && a.avgSpeed <= SPEED_THRESHOLD * model.maxSpeed(a.segmentId))
         a.derive( Hotspot, [confidence: Math.min(1, a.count/COUNT_THRESHOLD*0.5) ])
} }
```

Definition 3 Q1 and Q2 query specification

```
def q1 = new Query("speedsense.TrafficHotspots").area(
  minLat: 38.7379878, minLon: -9.1821318, maxLat: 38.758213, maxLon: -9.145832)
def q2 = new Query("speedsense.TrafficHotspots").area(
  minLat: 38.727875, minLon: -9.2002818, maxLat: 38.7683250, maxLon: -9.1276818)
```

3.2 Evaluation

The SpeedSense evaluation was performed in a custom simulation environment of a fixed and mobile node infrastructure. Fixed nodes are distributed randomly across the urban space with a minimum inter-node distance of 250 meters. A one-hop overlay network connecting the fixed infrastructure is simulated through a shared common peer database that provides a consistent view of the network membership. The network is static i.e., membership is determined on startup and there are no node entries or exits during execution, or node failures. Mobile nodes interact with a fixed-node homebase counterpart directly, resulting in the delivery of raw GPS data with no latency. Communication between fixed nodes experiences latency and jitter. Each mobile node simulates a vehicle, according to a traffic model, and reports its GPS reading every 5 seconds as it follows the assigned path. A common clock is used to timestamp readings; thus, any effects of clock desincronization are not considered.

Traffic is modeled by emulating a fleet of vehicles driving through random routes. The maximum speed for a given segment is the same value used for congestion detection and depends on the road type. An average speed, for each road segment at a given time, is determined by its car density (averaged over the previous 10 seconds) and used to generate the random speed individually for each vehicle, according to a normal distribution. In the experiments performed, congestion occurs in segments with a density of at least 5 vehicles. Vehicle paths are determined by choosing a random start position and sequence of road intersections. In order to induce traffic confluence, higher probability is given to highways and primary roads; a new path is computed whenever a vehicle reaches its destination. Figure 2 shows a rendering of the traffic simulation.

The simulation scenario covers the Lisbon urban area. This area is served by 500 fixed nodes and the mobile infrastructure comprises the same number of nodes. Two queries were tested, Q1 and Q2, covering respectively 6.25% and 25% of the overall simulation area (cf. Definition 3). Both were placed on a high mobile node density area. The set of metrics captured was averaged over 10 runs, corresponding to different fixed node placements, and compared to a centralized processing architecture, where GPS data is received and processed by a central node.

Workload Distribution One of the purposes of the experimental evaluation was to determine how the effort required to evaluate a query is spread among the fixed nodes. For that, workload is measured as the number of data acquisition and aggregation events occurring and processed at each fixed node. Specifically, the former pertains to the number of GPS sensor readings received and processed

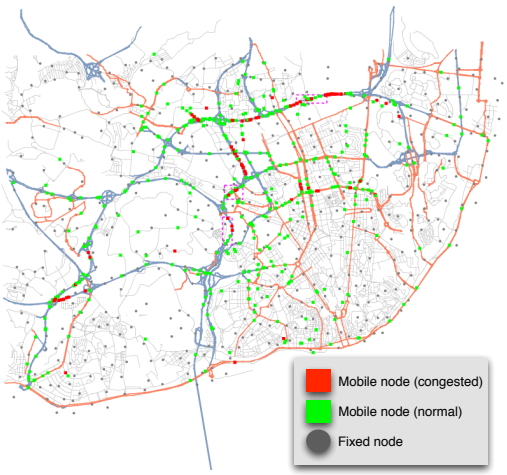


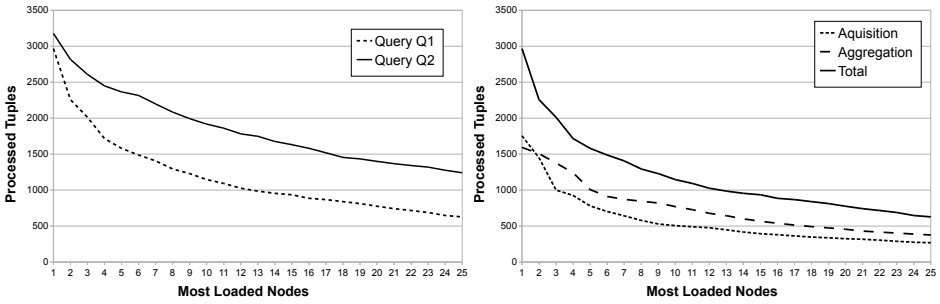
Fig. 2: Snapshot of the traffic simulation showing congested mobile nodes in red

by the acquiring node, while the latter refers to the number of inputs handled by the global aggregation stage and corresponds to the updates received for each segment aggregated by that node. To derive a total workload at each node, the two are added with equal weights.

Tabe 1 presents the workload obtained for Q1 and Q2 using a centralized approach, while Figure 3 shows the workload distribution for the same queries using the QTree distribution strategy.

Query	Acquisition	Aggregation	Total
Q1	17,541	6,448	23,989
Q2	44,634	16,493	61,127

Table 1: Workload for Q1 and Q2 using centralized processing

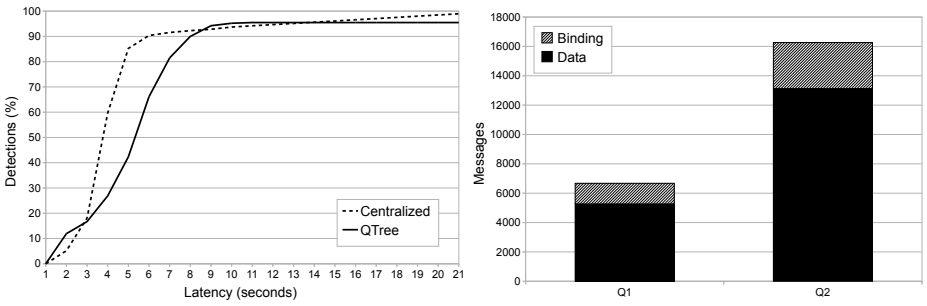


(a) (b)

Fig. 3: Workload using QTree (a) Total workload for query Q1 and Q2 (b) workload decomposition for Q1

In QTree, the most loaded node handles 7.4% and 3.1% of the total work for Q1 and Q2, respectively. The additional work introduced by Q2 is distributed among participating nodes and does not affect significantly the maximum workload. Relative to a centralized approach, the most loaded node in QTree handles 12.5% and 5.2% of the total workload processed by a central node for Q1 and Q2, respectively.

Query Success and Latency Query success is given by the percentage of accurate detections, including false negatives, where a false negative occurs when no detection is received within 120 seconds of occurrence. Detection latency times the lag between the occurrence of a segment congestion and the arrival of the respective detection at the query root. Transient congestions (lasting less than 20 seconds) were not considered for the evaluation. The results are only indicative, as the traffic patterns produced by the traffic model are highly dynamic compared to real world conditions, with several short lived congestions occurring during query evaluation. Figure 4a, which plots query success versus detection latency for both queries, shows a success rate in excess of 90% within the 120 second allowed window. Average query latency is higher relative to a centralized approach, which can be explained by the intermediate aggregation levels required by QTree.



(a) (b)

Fig. 4: (a) Query success and latency (b) Messages exchanged for Q1 and Q2 using QTree

Communication Load Cost measurements were also made regarding the number of messages exchanged during the execution of a query, providing an indication of the expected performance in terms of network usage. The communication load measures the overhead introduced by QTree relative to a centralized approach, including data messages and binding events. The former accounts for tuples exchanged between peers, relative to query data (incomplete aggregations that are forwarded up the aggregation tree) and query results (complete aggregations that are forwarded to the query root. While the latter capture the additional overhead associated with uploading the data from mobiles to a fixed node.

In all cases, the number of messages sent by individual nodes is limited at 1 message per pipeline stage every 10 seconds by the use of the *timeWindow*

operator, thus the exchanged messages reflect the number of nodes involved in query processing.

It was observed, as shown in Figure 4b, that the impact of the query area on message traffic is significant, resulting in an increase of around 140% in the total number of messages for a 300% increase of the query area.

4 Related Work

The availability of mobile devices with several sensors, such as smartphones, has lead to the creation of a large number of personal sensing applications, such as applications to record walks, routes, etc. These applications focus mostly on archiving and personal monitoring, for instance for health monitoring or fitness applications.

Some of these applications involve sharing among a specific community group or social network. In public sensing, data is open to the public at large. For example, in BikeNet [4], users can record their bike rides in their mobile phones and share this information with a community. The aggregation of this information, executed in a single server, allows community members to get information about the most popular bike routes.

CarTel [8] focus on vehicle based sensing applications. The system has a centralized architecture, where applications are hosted in a central server. Mobile nodes, executing in vehicles, collect the information needed by the running applications. Example applications include hot spot detection, and monitoring of road surface conditions [5].

Our approach differs from these systems in a number of ways, the most important being its decentralized architecture. A decentralized solution helps scalability by spreading the load and storage requirements by the participants in the system. Lacking the need for having a powerful server infrastructure also makes this approach more suitable for community-based sensing, with the needed resources being contributed by the community.

Some sensing systems present an architecture built entirely in the mobile nodes and ad-hoc coordination to support applications (e.g. [13] and [9]). A limitation of this approach is that mobile nodes have to spend computation, communication and energy resources in coordination efforts, regarding node and service discovery and context dissemination. This is specially relevant given that communication can be expensive and energy is scarce.

Other systems (e.g., [10,7]) have an architecture more similar to ours, based on the combination of fixed and mobile nodes. For example, in SensorWeb [7], a set of fixed nodes act as gateways for sensor network and proxies for mobile devices. In this system, a coordinator node mediates and coordinates the access of applications to the different gateways and proxies. Unlike SensorWeb, in the proposed architecture, nodes are also used to process information, thus allowing to more evenly distribute the load.

5 Final remarks

The proposed framework abstracts application developers from the complexity inherent to a distributed infrastructure, such as the actual location of relevant

data and the balancing of processing work, and supports common processing and aggregation tasks through a library of out-of-the-box components. Applications can share data through *virtual tables*, thus promoting the development of application mashups, while keeping control over the granularity of published data. Although the QTree distribution strategy is affected by the unbalanced distribution of sensed data, the strength of the strategy resides in the ability to limit the propagation of partial state by leveraging the partitioning of geographical space. Our initial evaluation shows that the proposed architecture can efficiently distribute the query processing load among nodes.

Directions for future research include the exploration of efficient delivery of query results to the mobile node base, and the optimized processing of multiple overlapping queries; finally we would like to investigate the aspects related to data persistence and historical queries.

References

1. W. Bloomin. http://whatsbloomin.com, June 2010.
2. A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn. The rise of people-centric sensing. *Internet Computing, IEEE*, 12(4):12–21, 2008.
3. D. Cuff, M. Hansen, and J. Kang. Urban sensing: out of the woods. *Commun. ACM*, 51(3):24–33, 2008.
4. S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. The bikenet mobile sensing system for cyclist experience mapping. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 87–101, New York, NY, USA, 2007. ACM.
5. J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. In *The Sixth Annual International conference on Mobile Systems, Applications and Services (MobiSys 2008)*, Breckenridge, U.S.A., June 2008.
6. P. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: an architecture for a worldwide sensor web. *Pervasive Computing, IEEE*, 2(4):22–33, Oct.-Dec. 2003.
7. W. Grosky, A. Kansal, S. Nath, J. Liu, and F. Zhao. Senseweb: An infrastructure for shared sensing. *Multimedia, IEEE*, 14(4):8–13, Oct.-Dec. 2007.
8. B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. K. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A Distributed Mobile Sensor Computing System. In *4th ACM SenSys*, Boulder, CO, November 2006.
9. N. Kotilainen, M. Weber, M. Vapa, and J. Vuori. Mobile chedar " a peer-to-peer middleware for mobile devices. In *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 86–90, Washington, DC, USA, 2005. IEEE Computer Society.
10. Y. J. L. Marie Kim, Jun Wook Lee and J.-C. Ryou. Cosmos: A middleware for integrated data processing over heterogeneous sensor networks. *ETRI Journal*, 30(5), October 2008.
11. Mohan, Prashanth and Padmanabhan, Venkata and Ramjee, Ramachandran . Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones. In *Proceedings of ACM SenSys 2008*, November 2008.
12. OpenStreeMap. http://www.openstreetmap.org/, April 2010.
13. O. Riva and C. Borcea. The urbanet revolution: Sensor power to the people! *Pervasive Computing, IEEE*, 6(2):41–49, April-June 2007.

Displaybook - Bringing online identity to situated displays

Abel Soares[1], Pedro Santos[1], Rui José[2]

[1]Mestrado em Informática, Universidade do Minho, Portugal
{pg13019, pg15964}@alunos.uminho.pt
[2]DSI, Universidade do Minho, Portugal
rui@dsi.uminho.pt

Abstract. This work is part of a study in which we aim to explore multiple bridges between on-line and off-line forms of socialisation by creating bi-directional connections between Facebook and situated social interactions. In this paper, we specifically describe a study on the use of public displays for the public presentation of data from the Facebook profiles of people near the display. The key challenge is how to map the concept of sharing information within a social network, to the concept of sharing information with the places you visit. For this to be viable, people must have full control over what they share and in what circumstances they will share it. This paper addresses this issue by studying the sharing alternatives, how this sharing of profile data in a public display is perceived by people and what are the main factors affecting that perception. The results suggest that, overall, people seem to be willing to expose parts of their Facebook profiles if given proper privacy controls. However, the study has also revealed a clear gap between privacy control in Facebook and the type of privacy controls that would be needed for this particular use of Facebook information.

1 Introduction

With their increasingly huge popularity, Social Network Sites (SNSs) have been reshaping many notions of socialisation. These sites are essentially about people wanting to stay connected with their friends and other people around them. This normally involves sharing with those people information regarding multiple facets of their lives. By feeding their social profile, posting content, expressing feedback and commenting on the activity of others, people are continuously generating massive quantities of user generated content that is quickly disseminated through the user network. All this activity is inherently on-line, occurring in the web almost as a parallel world of relationships and interactions that may seem independent from the situated interactions of the physical world. However, research has shown that Facebook connections normally have a strong correlation with off-line proximity [1][2]. More than a way to meet new people, SNSs are mainly a new mechanism for

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 403–414

managing already existing connections, albeit weak ones. Many of these "weak ties", i.e. relationships with people outside the normal groups of which we are a part of, emerge from the existence of a common offline element, such as working in the same place, studying at the same college, or frequenting the same places. The role of these weak ties in SNSs is well-know, but the role of SNSs in the off-line interactions between those people has been less explored, meaning that there is still a strong potential in extending SNSs to situated interaction and presence. This might be particularly important to the process of meeting new people, something that SNSs still do not seem to afford very effectively [2].

1.1 Bridging between on-line and off-line social interactions

This work is part of a study in which we aim to explore multiple bridges between on-line and off-line forms of socialisation by creating bi-directional connections between SNSs and the situated social interactions occurring between sets of co-located people. In this paper, we specifically describe a study on the privacy implications for the use of public displays to present data from the Facebook profiles of people near the display.

The motivation behind this work is the idea that, in some situations, bringing SNSs profiles into the off-line world of situated interactions may enrich those interactions. SNSs profiles may provide a relevant, yet spontaneous and informal, representation of identity that can be useful for many types of situated services. Public displays, in particular, may provide an additional channel for self-exposure and new opportunities for augmenting co-located social interactions. Also, from the perspective of SNSs, this type of bridge may lead to a stronger presence in the life of their users, something that may turn out to be crucial to their long-term sustainability.

However, the use of Facebook profiles outside their normal context also raises considerable challenges, both technical and social. The key challenge is clearly how to map the concept of sharing information within a social network, to the concept of sharing information with the places you visit. For this to be viable, people must have full control over what they share and in what circumstances they will share it. This paper addresses this issue by studying the sharing alternatives, how this sharing of profile data in a public display is perceived by people and what are the main factors affecting that perception.

To support this study we have developed Displaybook, an application that enables people to share Facebook profile information with public displays. The information from a particular profile is only presented when that person is detected near a display. This achieved by associating a Bluetooth address with each profile and scanning Bluetooth devices in the vicinity of the display. When installing the application,

people are given the opportunity to choose between a set of privacy preferences. We have collected data on the privacy specifications selected by people and we have conducted interviews with some of the users to gain a more in-depth perspective on individual perceptions. The results suggest that overall people seem to be willing to expose parts of their Facebook profiles given proper privacy controls. However, the study has also revealed a clear gap between privacy control in Facebook and the type of privacy controls that would be needed for this particular use of Facebook information.

2 Related Work

The use of SNSs within multiples situations of everyday life is increasingly possible through all forms of mobile versions of the respective software, allowing people to continuously maintain their on-line presence and possibly generate life streams representing their off-line social activity. However, these mobile applications do not really take advantage of the opportunity to connect the social context of SNSs with the social context of co-located people interacting with each other.

One of the early experiments in bridging between social networks and social situations, more specifically a conference setting, was proposed by Konomi et. al. in [3]. The conference badges were RFID tags and when participants approached a display, a social network was presented base on publication records in DBLP. The expectation was that presenting the professional social network within that specific context would help co-located participants to communicate and develop relationships.

The injection of real-world presence data into social networks is explored in CenceMe [4]. A mobile application is continuously collecting data from any sensors the mobile phone might have and inferring the current activity of the person. This data can then be injected into multiple social networks, including Facebook, thus enriching the information flow from the real-world to the SNSs. CenceMe also leverages on the social networks for defining access policies. It basically takes the buddy lists users have already created in those services to determine who can have access to the CenceMe data.

Kostakos has conducted a study in which Facebook connections of 2602 individuals were analysed in conjunction with data about their Bluetooth encounters [2]. This study provides important insight into the multiple similarities and shared connections between these two types of social structures. This work also involves the use of a Facebook application and Bluetooth, but in this case they are being used as data collection tools for studying the social networks themselves. In a separate piece of work within CityWare, Kostakos and O'Neill have also explored the use of a

Facebook application in association with Bluetooth traces [5]. In this case, Bluetooth mobility traces were presented in public displays, but the system also allowed people with the Facebook application to have access to data about their physical co-presence with members of their social network.

WhozThat [6] uses the SNSs profiles of people nearby to create context information that can then be used to support spontaneous interactions or drive the music selection. People are expected to use a mobile phone running an identity sharing protocol that will advertise their on-line identities to the other nearby devices. This system does not consider the use of public displays or any explicit selection of which information to share, but it is an example of using SNSs profiles as a sort of personal data aura that can be used to mediate digital self-exposure.

Bohmer and Muller [7] conducted a study on the exhibition of SNSs profiles in public settings. Using mockup images they asked people about their willingness to expose profile information in two types of what they called social signs. The first was a personal social sign projected around the person and showing parts of the respective profile. The second was a interpersonal sign, projected in such a way to link two people and representing some type of connection between them, such as having a mutual friend or sharing an interest. The study consisted in presenting the scenarios and interviewing people about their perception of those hypothetical uses. The results suggested that there is some interest in the overall idea, but also highlighted serious concerns about the particular circumstances in which such exposure should occur. This study, however, did not address real usage situations or how to map particular types of self-exposure to specific situations.

3 System Overview

3.1 Mapping Facebook concepts into public displays

A key part of this work is to study how to map Facebook concepts to the needs of publication in information displays. Regarding which information to take from Facebook, we have studied the existing information and how it can be accessed. In principle, any piece of information and activity generated by a Facebook Profile could be shown on the public display, as long as the respective user had authorised access to that information to our Displaybook application. However, information presentation in public displays represents a very unique context for exposing personal information. Therefore, individuals should have a clear opportunity to specifically express permissions to that particular form of presentation. We have focused specifically on profile information, making a separation between public information (name and photo) and non-public information (Likes, music, TV, movies, books, quotes, About

me, Activities, Interests, Groups, Events, Notes, Birthday, Religious and political views, Education History, Work history and Facebook status).

The information sharing model in Facebook was conceived with a rather different set of assumptions, and thus it was without surprise that we have identified a number of mismatches when trying to map that model into a model for presenting data in public displays. More specifically, the following issues were identified:

- Facebook privacy model defines how information in Facebook can be accessed, in this case by applications. It does not really support any type of considerations on how the information is going to be used. This basically means that before we present users with our own application-specific privacy settings, they will first have to go through the standard privacy settings in the Facebook platform, where they will have to authorise information access regardless of how it will be used by Displaybook.
- Facebook applications are mainly designed to be used interactively, and thus the basic permission model only allows applications to access user's data when the user is logged in. There is an alternative to overcome this limitation in which users may allow applications to access their data without the need of an active session. This is however not a common need for most Facebook applications and represents an additional step for privacy-sensitive users.
- The user can easily specify its privacy preferences for an application, but it is normally very easy for an application acting on behalf of the user to gather data about friends of users (and friends of friends), even if they are not aware of this information usage. This is a reflection of the Facebook network and privacy model, in which friends normally have the ability to access much more data than "anyone".

To summarize, and because of these mismatches, simply installing Displaybook seems to entail many more privacy risks than what it would seem necessary when we consider the information actually presented on the public displays. Regardless of the information actually being presented, regardless of any blurring or aggregation mechanisms that might be used to preserve privacy, the fact remains that to be usable, Displaybook will always end-up having considerably extensive permissions to access Facebook profiles.

3.2 The Displaybook application

As part of this study, we have developed the Displaybook application for enabling the presentation of parts of Facebook profiles in public displays. The application is composed by 4 key components, as represented in Fig. 1.

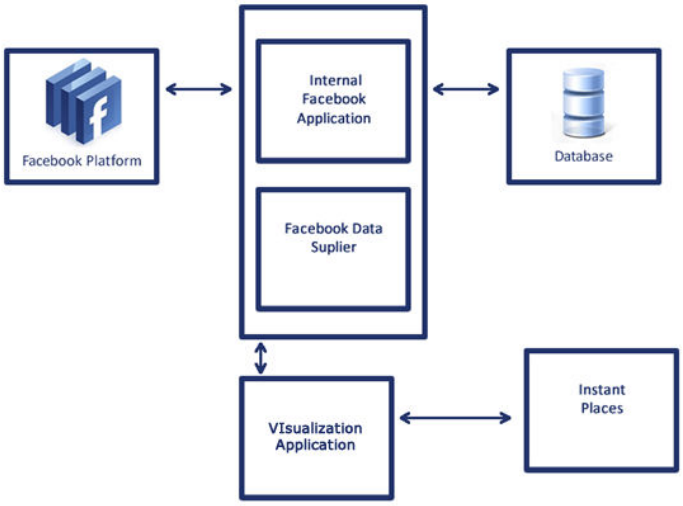


Fig. 1. – System overview.

Internal Facebook Application – This is an application that runs as a Facebook application and can be accessed from inside Facebook. It serves as an integration point for users, allowing them to specify privacy settings for data on displays and associate their Bluetooth MAC address. Users of our system must necessarily take the step of installing this application from their Facebook account.

Facebook Data Supplier – This is a web application that uses the Facebook API to retrieve the necessary data about the profiles that are using Displaybook. The data supplier will receive a set of MAC addresses (currently detected in the place) and will get all the Facebook data associated, keeping in mind the permissions that users have set.

Visualization application – This application supports two types of Flash-based visualizations of the Facebook profile data. The first visualization, represented in Fig. 2, displays the list of present profiles. Each profile is represented by an icon that may include the name and the photo, unless the user has denied any of this information. The second visualization, represented in Fig. 3, displays an aggregate view of the information from the multiple users detected in the place. This information may include gender, birthday, location, relationship status, hometown, education and high school education and is presented without being associated with any particular profile. When data is similar for users, the tag gets more relevance, with a superior size compared to others.

Fig. 2. – Profile visualization application.

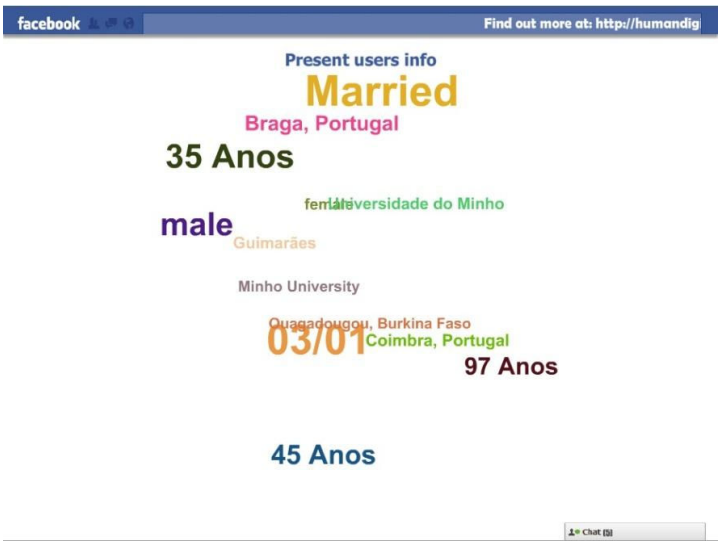


Fig. 3. – Aggregate visualization application.

Instant Places – Instant Places is a Bluetooth based sensing platform and provides information about open Bluetooth presence sessions. A session symbolizes a presence of a Bluetooth enabled device in the place. This information is obtained from routers running Bluetooth scanning software.

When the system is running, the visualization application queries Instant Places and obtains a list of the Bluetooth devices that are currently present near the display. The application will then query the Facebook data supplier for data associated with those Macs. The Data Supplier will find correspondent Facebook Profiles IDs and their permissions settings in our database and use Facebook Platform API's to access data about the users. After that, the Data Supplier will filter all data according to each user permission setting and respond back with the data to the Visualization Application in XML format.

3.3 Setting privacy policies with Displaybook

When a user first goes to the Displaybook application, he or she must go through a set of dialogs for setting privacy preferences. The first dialog is meant to give Displaybook permission to access the public profile data. This is a common procedure when someone uses a Facebook application and is absolutely necessary for the system to be able to present parts of the profile.

Fig. 4. – Request for permission dialog.

Immediately after, the user will be asked to give offline access and permission to some of his or her profile information. This will make possible for Displaybook to

retrieve Facebook Profiles information on behalf of the user. It also indicates the profiles categories that the system may ask for, such as Birthday and Education, Hometown and Location, and Relationship Status. This is a generic permission from Facebook. Users will later be asked within the application to refine exactly which information they want to display.

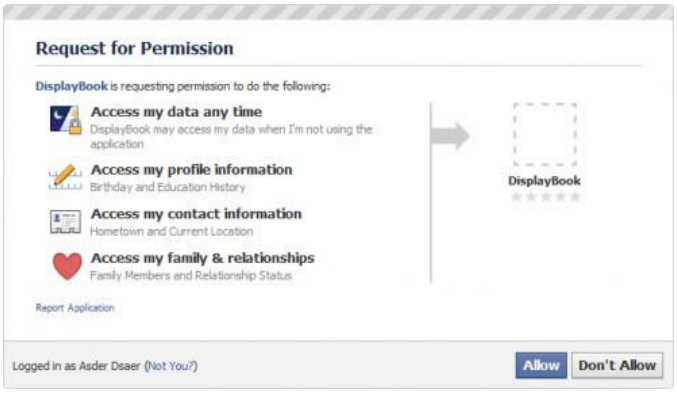


Fig. 5. – Extended Permissions Dialog.

In the next step, people are asked to submit their mobile device Mac address. This will later be used to enable the displays to recognize the presence of the user.

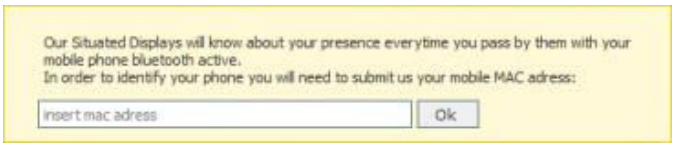


Fig. 6. – MAC address association.

Now that Displaybook already has access to the Facebook information, the permissions dialog can be used to configure exactly which profile data users have interest in showing on the displays. The answers to this dialog were an important part of our study.

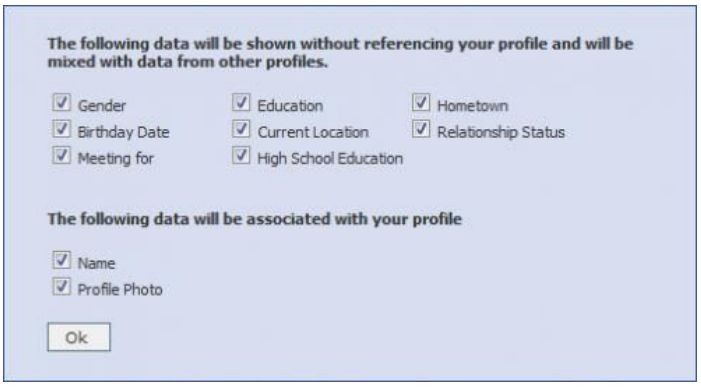


Fig. 7. – Permissions Manager.

4 Evaluation

4.1 Deployment

We have used two public displays in our department as the basis for the deployment. This is a place closely linked with a particular community, the Department staff and students. Many of them share Facebook connections, and most people could easily be informed about the system and invited to join in.

The content generated is a simple web-based application. The respective URL is given to the display software, which will show that content whenever requested. Bluetooth sensing is in place to support the presence recognition and also to support explicit interaction using a simple command language for the Bluetooth Names.

The announcement of the applications was made by sending an e-mail to the local e-mail lists and through Facebook itself. The announcement included an indication of where to install the Facebook application, the privacy policy and usage information, especially how to create the connection between Bluetooth Mac addresses and the identity profile.

4.2 Results

There were nine users using Displaybook during the one week long evaluation period. They were all part of the Department community, including academic staff, researchers and administrative staff. Eight of them have used the privacy manager to change their data permission settings. The data collected on how they set their privacy settings can be summarized as follows:

- **Name:** No one has changed permissions for name. It was always available, for all users, through the time the experience occurred.
- **Profile photo:** Three persons have removed permissions to show the photo. Two of them have done so when first confronted with the privacy dialog, and one after having seen her photo on the display.
- **Education:** Two persons have removed permissions to show education data. One initially, one after a while.
- **Birthday:** Three persons have denied presentation of their birthday.
- **Relationship Status:** Five persons have blocked permissions to show their relationship status.
- **Location:** Everyone has allowed presentation of their location (Home Town)

Users who have blocked the availability of the profile photo on the public displays, have also expressed that they don't really feel comfortable about having their identity displayed in public spaces. One user has reported having an unpleasant experience when people recognize him and approached him to talk about it. This kind of confrontation does not exist in a virtual presence and clearly shows the type of conflict that may occur between what people feel and construct about their virtual existence, and what happens when that same virtual identity suddenly gains a situated existence in physical space. This particular example, and also some of answers in the interviews, have shown that names and photos were seen as particularly sensitive information when presented in this context. This reveals a clear contradiction between what Facebook regards as public data in an online environment (name and photo) and what people would be more willing to show on a public display.

4.3 Lessons Learned

One of the lessons from this work has been to feel as developers the extent of what it means to say that Facebook is an evolving reality. If considering that this work has been conducted within a three months period, we still had to face changes in the underlying API, in the application naming policies, in the concept of "fan page" (now a Facebook page liked by people), and most importantly multiples changes in the privacy policies and controls. More than presenting these problems as complaints, we expected to highlight how important it might be for Facebook developers to reduce their exposition to changes in the Facebook API or even in the application and privacy policies.

The way we had to present the privacy settings has revealed a clear mismatch between Facebook assumptions on the use of profile information and our own use of that information. Facebook assumes that either information is shared or is not shared, which is probably a reasonable assumption within the normal Facebook setting. However, within the setting of our study, it was clear from the beginning that the way in which information was presented and even the circumstances surrounding its

presentation would be important elements in how people perceive that their privacy is being affected.

5 Conclusions

The use of Facebook for creating user-generated content on public displays clearly holds a lot of potential. However, control by users is crucial in such approach, and this work has clearly shown how Facebook privacy policies are not aligned with this particular usage of Facebook data. When bringing Facebook Profiles into public displays, privacy concerns enter a new dimension that is not necessarily the dimension exposed in web environments, where typically social platforms exist. The identification of users is a concern in public spaces, something that is normally not a major issue in SNS. As future work, we intend to study how privacy policies for self-exposure in public displays can be expressed more effectively by users, and also how users can take more advantage of these features as a mechanism for situated interaction.

References

[1] N. Ellison, C. Steinfield, e C. Lampe, “The benefits of Facebook "friends:" Social capital and college students' use of online social network sites.,” Journal of Computer-Mediated Communication, vol. 4, 2007.

[2] V. Kostakos, “An empirical study of spatial and transpatial social networks using Bluetooth and Facebook,” 0910.4292, Out. 2009.

[3] S. Konomi, S. Inoue, T. Kobayashi, M. Tsuchida, e M. Kitsuregawa, “Supporting Colocated Interactions Using RFID and Social Network Displays,” IEEE Pervasive Computing, vol. 5, 2006, pp. 48-56.

[4] A.T. Campbell, S.B. Eisenman, K. Fodor, N.D. Lane, H. Lu, E. Miluzzo, M. Musolesi, R.A. Peterson, e X. Zheng, “CenceMe: Injecting Sensing Presence into Social Network Applications using Mobile Phones (Demo Abstract).”

[5] V. Kostakos e E. O’Neill, “Capturing and visualising Bluetooth encounters.,” CHI 2008, workshop on Social Data Analysis, Florence, Italy.: 2008.

[6] A. Beach, M. Gartrell, S. Akkala, J. Elston, J. Kelley, K. Nishimoto, B. Ray, S. Razgulin, K. Sundaresan, B. Surendar, M. Terada, e R. Han, “WhozThat? Evolving an ecosystem for context-aware mobile social networks,” Network, IEEE, vol. 22, Jul. 2008, pp. 55, 50.

[7] Matthias Böhmer e Jörg Müller, “Users' Opinions on Public Displays that Aim to Increase Social Cohesion,” Proceedings of The 6th International Conference on Intelligent Environments. Kuala Lumpur 2010, Malaysia; to appear.

Novos Serviços Turísticos para Mobile Advertising

Leonel Dias[1,2] e António Coelho[1,2]

[1] DEI/FEUP, R. Dr. Roberto Frias, 4200-465 Porto, Portugal
[2] INESC Porto, Campus da FEUP, R. Dr. Roberto Frias, 4200-465 Porto, Portugal
ei05041@fe.up.pt,acoelho@fe.up.pt

Resumo Actualmente, a computação móvel disponibiliza novas capacidades que potenciam o desenvolvimento de novos serviços. Geralmente as fontes de informação são heterogéneas, dispersas e associadas a localizações geográficas. Para além da localização e contextualização com o perfil do utilizador, os sistemas de computação móvel estão a sofrer evoluções, para suportar funções ligadas à publicidade e marketing de bens e serviços. Assim, neste trabalho apresenta-se uma solução genérica para a disponibilização de novos serviços de publicidade e marketing, para o sector do turismo que tomem partido do contexto espacial do utilizador – *Mobile Advertising*. A implementação prática da metodologia sugerida, é focalizada na Região de Turismo do Douro.
Palavras-chave: Computação móvel, *Mobile Advertising*, Informação geográfica e Serviços baseados na localização

1 Introdução

A contribuição do Turismo para o Produto Interno Bruto (PIB) nacional está situado nos 14.4% e a World Travel Tourism Council (WTTC) prevê que em 2020 seja 16.9% [1]. O Turismo é também responsável por 18.8% da empregabilidade nacional. O plano tecnológico nacional português [2] contempla várias estratégias na utilização das novas tecnologias para criar valor acrescentado na indústria do turismo. É necessário que as entidades turísticas pautem as suas actividades por parâmetros estratégicos, tais como, Marketing mais agressivo e directo; Inovação na comercialização dos seus produtos; Intensificação dos contactos personalizados para promoção de serviços; E aposta nas novas tecnologias para promover a aproximação dos diversos interlocutores.

Destas políticas nasce a importância para o desenvolvimento de serviços e plataforma inovadoras, que permitam ao sector do Turismo crescer e potenciar as suas mais-valias. O *Mobile Advertising* é o resultado dos princípios propostos pelos conceitos de Publicidade, Marketing e GeoMarketing, sendo como o próprio nome indica uma forma de fazer ou transmitir publicidade através de aparelhos móveis. A principal potencialidade do *Mobile Advertising* é a capacidade de promoção de campanhas com acesso directo ao utilizador ou conjunto de utilizadores em apenas alguns instantes. [3]

Este trabalho aborda a concepção de uma plataforma inovadora para *Mobile*

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 415–426

Advertising, que utiliza as tecnologias actuais para aproximar as pessoas das melhores regiões de turismo. Para demonstração da exequibilidade dos conceitos e mecanismos propostos é apresentado um caso prático de aplicação à Região de Turismo do Douro.

2 Trabalho relacionado

Nos trabalhos apresentados em [4], [5], [6] e [7], são descritas várias soluções para o desenvolvimento de soluções fornecedoras de serviços baseados na localização. São apresentadas várias alternativas não intrusivas permitindo ao utilizador a subscrição e recuperação da informação específica que lhe interessa, de acordo com as suas preferências e a sua localização actual. Tal como nestes trabalhos, o artigo [8] explora algumas ideias de como criar uma plataforma que modele, observe, avalie e explore uma boa noção do contexto presente.

Em [9], desenvolve-se um novo mecanismo para selecção de informação georreferenciada. A solução apresentada garante que a informação seleccionada tem em conta a orientação, sentido, ângulo e campo de visão do utilizador. Esta abordagem permite realizar interrogações do género: "Quais são os restaurantes que estão nesta direcção?". Recentemente foi também apresentado um outro trabalho, que conta com a utilização de elementos multimédia na pesquisa interactiva de pontos de interesse baseados na localização e orientação do utilizador [10]. Para tal, os dispositivos móveis são equipados com câmaras digitais e sensores de posição e direcção, servindo de complemento para a apresentação de pontos de interesse sobre um mapa. Esta abordagem ajuda a estabelecer a correspondência entre os ícones representativos sobre o mapa e os objectos reais do espaço físico que rodeia o utilizador.

No Geotumba, um motor de pesquisa geográfica para dispositivos móveis, os autores descrevem os principais desafios na concepção de interfaces para estes serviços em dispositivos móveis, para além de que criam métodos para definição, recuperação e visualização de informação de contexto geográfico [11]. Para estudar as mais-valias de interfaces de pesquisa que utilizam chaves em contextos móveis, um estudo experimental apresenta os resultados da pesquisa sobre mapas e da pesquisa sobre texto [12]. Os resultados mostram que a escolha depende de três factores: preferências pessoais, necessidade de informação e contexto da situação. O estudo conclui que uma solução híbrida é a melhor escolha para desenvolvimento de interfaces de pesquisa sobre informação georreferenciada.

Quando se pretende a visualização de informação sobre mapas, existem alguns problemas que podem ocorrer devido a: congestionamento de elementos num espaço limitado; detalhes de visualização que dependem da resolução do ecrã; e elementos que se tornam imperceptíveis a partir de uma determinada dimensão. Este tema é bastante relevante, quando se tratam de aplicações que se destinam a equipamentos móveis, cuja capacidade de memória, processamento e resolução é limitada. Assim em [13], [14], esta temática é explorada e são propostas algumas soluções ao nível de visualização, sendo dada especial atenção à utilização de mecanismos de filtragem baseados em funções de grau de interesse e

à definição de múltiplas representações que resolvem a densidade de informação e o grau de interesse do utilizador. Mas a visualização dos mapas não pode estar sempre dependente da ligação à internet, por isso em [15], apresenta-se uma técnica para armazenamento dos *tiles* de mapas, em memória secundária dos dispositivos móveis, que rompem com os métodos tradicionais de *caching*.

3 Plataforma para Mobile Advertising

O *Mobile Advertising* proporciona oportunidades para novos serviços. Tendo em conta as limitações das soluções existentes actualmente, desenvolveu-se uma nova plataforma para a divulgação de serviços ligados ao turismo regional.

Assim sendo, cada turista dispõe de uma aplicação móvel, que lhe permite explorar e conhecer uma determinada região, possibilitando também às organizações a divulgação dos seus serviços, limitando a interferência nas actividades dos utilizadores. O funcionamento principal da plataforma foi optimizado para ser simples e eficaz. Como está definido na figura 1, cada organização local possui pontos de interesses, aos quais podem estar associados eventos de várias competências. Estes eventos são publicitados através de um portal de gestão, onde existem as funcionalidades necessárias para a realização desses objectivos. Paralelamente, cada turista da região pode utilizar a aplicação móvel que lhe permite aceder ao menu de navegação, onde pode visualizar um mapa interactivo da região. Neste são identificados todos os pontos de interesse, de acordo com o perfil definido por cada utilizador, sendo visualizada toda a informação relevante relativa às suas características. Para além disso, beneficia de facilidades que lhe permitem visualizar itinerários, efectuar reservas antecipadas ou comentar esses locais através das tecnologias associadas às redes sociais actuais.

A cada reserva do utilizador fica associado um código, que deverá ser fornecido à organização à qual diz respeito a reserva efectuada. A validação deste código permite às organizações comprovar as reservas dos eventos lançados. Além disso, cada serviço que seja potenciado por uma pré-reserva, permite aos utilizadores acumular pontos de participação, que posteriormente podem ser convertidos em prémios e aos quais está associada uma determinada quantidade de pontos acumulados. Existe ainda uma outra forma para garantir a acumulação de pontos, que é realizada através do sistema de navegação, que assinala no mapa locais georreferenciados onde existe um *QR Code* disponível para ser descodificado e que garante que, após a sua conversão, é acumulado novamente um determinado número de pontos. A atribuição dos pontos não é homogénea, sendo que o seu valor unitário depende da política da administração, que terá em conta vários factores, destacando-se sobretudo o potencial de negócio e a influência da organização. A selecção de uma boa política que garanta excelentes benefícios económicos para as organizações, pode assegurar sustentabilidade financeira ao portal. Por exemplo, se uma organização local recebe mensalmente 250 reservas através da plataforma, a administração local pode celebrar com essa organização, um contracto que lhes permita aceder a 1% dos valores monetários

rentabilizados. A utilidade dos códigos está associado à validação das reservas e atribuição de prémios, que permite garantir a fidelização dos turistas na região.

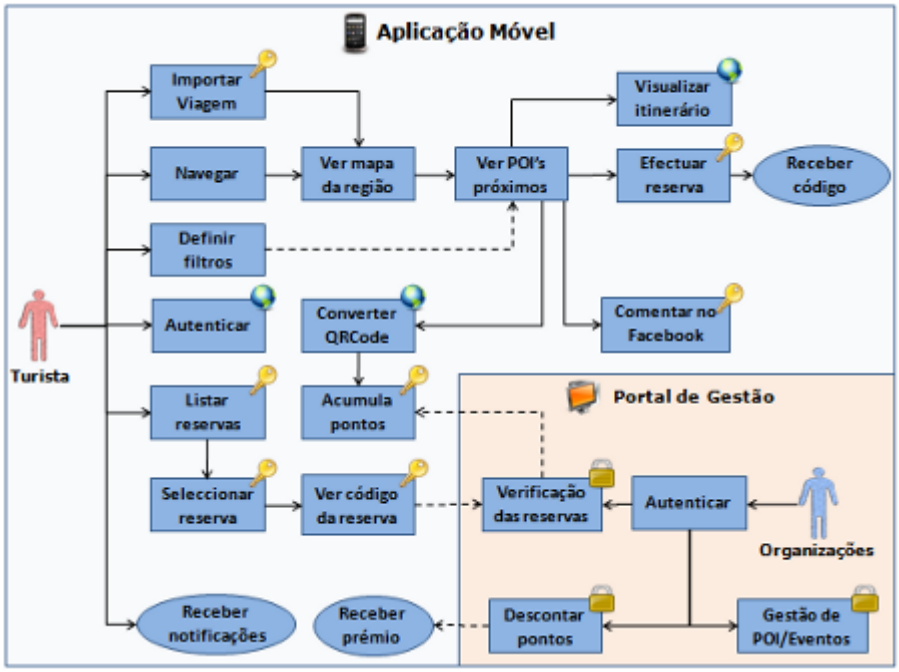


Figura 1. Funcionalidades principais

Existe também uma outra forma para as organizações publicitarem as suas actividades, que funciona através de notificações que são recebidas sempre que o utilizador se encontra suficientemente próximo de eventos patrocinados. Estas notificações estarão sempre de acordo com o perfil do utilizador e podem ser recebidas em qualquer circunstância desde de que o utilizador tenha permitido tal funcionalidade. Esta abordagem pode ser combinada com o mecanismo de conversão dos *QR Codes*, proporcionando às organizações a publicitação de vários pontos de interesse que permitem aos utilizadores acumularem pontos e onde, posteriormente nas redondezas, estão associados alertas de proximidade que lançam notificações de eventos próximos.

Para além destes mecanismos vocacionados para a implementação de Mobile Advertising em regiões turísticas, o utilizador tem ao dispor várias funcionalidades que lhe servem de guia turístico, garantindo que não instala somente uma aplicação para procura de serviços ou eventos promocionais.

A utilização de aplicações móveis, com acesso a dados remotos via internet, torna-se extremamente dispendiosa, sendo que a sua praticabilidade em certas

zonas do país ainda não é possível a 100%, uma vez que nas localidades mais remotas o seu acesso, utilizando dispositivos móveis é muito limitado. Por estas razões, a aplicação móvel beneficiará de dois modos de execução: *online* e *offline*. No módulo *offline*, isto é, sem acesso à internet, apenas são exequíveis as funcionalidades cuja existência de acesso à rede não é obrigatória. Neste módulo, a visualização de mapas é possível, sendo que a indicação dos pontos de interesse é efectuada de acordo com os dados mais recentes e que se encontram armazenados localmente no dispositivo. Para além disso, o utilizador pode continuar a receber notificações, sendo que estas dependem sempre dos dados armazenados localmente e que podem não estar actualizados. A partir deste ponto, não é possível continuar a utilização sem ser no modo *online*. Neste segundo modo, o utilizador pode-se autenticar no sistema, permitindo aceder ao seu perfil definido no portal Web da região. Para além disso, a autenticação permite o acesso aos serviços ligados ao *Mobile Advertising*, designadamente às reservas e aos pontos de participação.

3.1 Sistema Modular

Para implementação do conceito anteriormente descrito, decidiu-se estruturar a plataforma em 3 componentes principais. Como se pode observar na figura 2, existe um Portal Público, um Portal de Gestão e uma Aplicação Móvel. Cada um destes módulos possui funcionalidades próprias, sendo que o Portal de Gestão pode ser manipulado pela Administração do sistema e pelas várias Organizações locais existentes na região, enquanto que, o Portal Público e a Aplicação Móvel apenas terão como actor principal, os potenciais turistas.

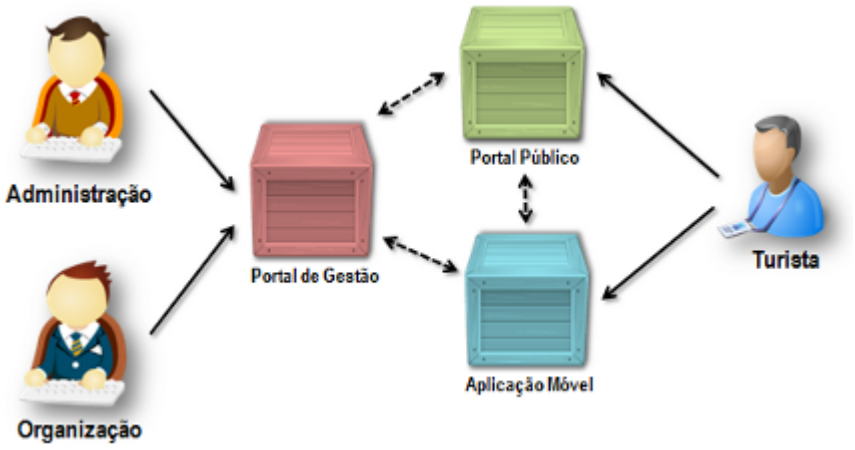


Figura 2. Módulos e respectiva interacção

O Portal de Gestão não é de acesso público, sendo facultado somente às organizações locais e à administração da plataforma. Neste pacote constam as funcionalidades para gestão de conteúdos presentes no Portal Público, administração das contas dos utilizadores, configuração dos pontos de interesse e eventos associados, bem como todas as funcionalidades ligadas à opção de reservas.

O Portal Público é de acesso geral, sendo o núcleo da plataforma, pois permite várias facilidades tais como o acesso à informação alfanumérica e multimédia, sempre que possível georreferenciada (exemplo, noticias e publicidade de eventos ou serviços), e a manipulação de aplicações geográficas que permitem ao utilizador interagir com os conteúdos georreferenciados e a possibilidade para programação de viagens ou passeios na região turística.

A Aplicação Móvel pode ser analisada como a extensão do portal público que permite à plataforma ficar mais próxima do utilizador, devido à sua característica móvel e à sua capacidade de contextualização geoespacial. Esta componente implementa todos os mecanismos para *Mobile Advertising*, bem como outras valências que criam valor acrescentado no uso global da plataforma. É composto pelos seguintes módulos:

- Exploração – Conjunto de funcionalidades que permitem navegar sobre uma região, através das facilidades geográficas de visualização e localização de pontos próximos à localização do utilizador;
- Publicidade – Serviços que operam sobre os eventos submetidos pelas organizações e que são oferecidos aos utilizadores, consoante o seu perfil e o contexto das suas acções. É também responsável pelos serviços que permitem a acumulação de pontos;
- Reservas – Permite efectuar reservas antecipadas, para obtenção de descontos e pontos de participação. Implementa também os mecanismos necessários à gestão das reservas, nomeadamente a visualização dos códigos comprovativos, que permitem fazer a respectiva validação;
- Redes Sociais – Implementa as funcionalidades responsáveis pela autenticação utilizando as contas das redes sociais e de inserção de comentários, relativos a pontos de interesse previamente seleccionados pelo utilizador, garantindo que estes são colocados em tempo real e associados à sua localização.

3.2 Arquitectura

A arquitectura da plataforma é composta pelos componentes que permitem garantir a implementação dos serviços ligados ao *Mobile Advertising*. Para além disso é necessária a introdução de outros componentes que facilitem o desenvolvimento e integração das funcionalidades complementares aos objectivos primários do sistema global.

Assim sendo, esta plataforma integra um servidor que possui uma base de dados espacial contendo informação alfanumérica e geográfica. Esta base de dados está acessível pelas diferentes componentes, através de um Web Service que permite a manipulação e o acesso à informação aí existente.

Figura 3. Arquitectura da plataforma

Este servidor disponibiliza as interfaces necessárias para a implementação do portal de gestão e do portal público acessíveis via Internet através dos browsers instalados nos diferentes sistemas operativos. Ambos os portais apresentam mapas interactivos para apresentação e manipulação da informação georreferenciada, por isso é necessária a existência de um servidor de mapas. Após alguns testes efectuados, foi decidida que a melhor abordagem seria a utilização de um servidor de mapas externo, pois permite poupar tempo de desenvolvimento assim como reduzir custos, uma vez que na sua maioria são de acesso gratuito. Outra vantagem associada aos servidores de mapas externos é que estes normalmente incluem API's poderosas e que se encontram em constante actualização, permitindo óptimas performances e excelente usabilidade. A tecnologia de servidor de mapas seleccionada para a implementação desta metodologia foi o Google Maps API V3. No entanto, caso uma determinada implementação exija a utilização de outro servidor de mapas externo, por exemplo por motivos de protocolos estabelecidos, a sua substituição não será crítica, uma vez que cada componente é desenvolvido, de forma o mais abstracta possível, relativamente à tecnologia utilizada pelas outros componentes.

Para aumentar as potencialidades da aplicação, foi decidido integrar algumas funcionalidades oferecidas pela redes sociais. Assim sendo, todos os mecanismos desta área, utilizam o Facebook e respectiva API, uma vez que se trata de uma das mais importantes redes sociais. Nesta arquitectura há ainda lugar para a componente móvel, essencial para a implementação dos conceitos ligados ao

Mobile Advertising e *Location-Based Services*. Este modelo pode ser aplicado a qualquer sistema operativo móvel, mas neste trabalho a implementação foi desenvolvido para Android, de forma a beneficiar das facilidades oferecidas por este sistema, nomeadamente para os mecanismos ligados à visualização geográfica e geo-localização. O dispositivo móvel dispõe de uma base de dados SQLite, que é instalada de raiz com o sistema Android e que será utilizada para armazenar toda a informação que é necessária manter localmente. A localização do dispositivo é efectuada através do GPS, sendo por isso independente do acesso à rede de internet.

4 Caso de estudo - MOBIDouro

Para implementação da plataforma, considerou-se que a região do Douro seria uma boa oportunidade para ser o primeiro caso de estudo da solução desenvolvida. Para implementação desta solução apenas se descrevem os mecanismos ligados à aplicação móvel e ao desenvolvimento do sistema, uma vez que relativamente ao portal Web, este está a ser desenvolvido no âmbito de outro projecto de investigação.

4.1 Aplicação Móvel

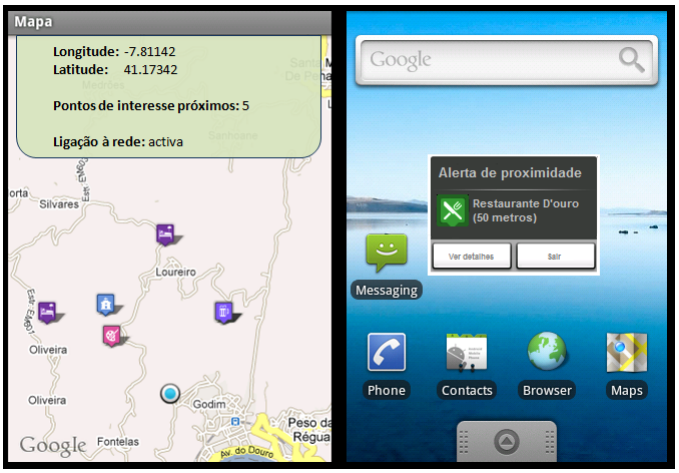


Figura 4. Exemplos de interface

A aplicação móvel possui um conjunto de funcionalidades relacionadas com a opção de exploração geográfica, onde o utilizador tem acesso a um mapa dinâmico contextualizado com a posição actual do utilizador. Sobre este mapa

são indicados os pontos de interesse que se adequam ao perfil definido pelo utilizador e para cada ponto assinalado é possível visualizar com detalhe, tando a informação alfanumérica, como todos os eventos associados a este. Nos eventos em que seja possível fazer uma reserva, é fornecido ao utilizador um formulário que lhe permite efectuar essa operação. Sempre que a aplicação se encontra em execução, podem aparecer alertas de proximidade no visor do dispositivo móvel, que informam o utilizador de eventos muito próximos (distância inferior a 250 metros) e cujas características também se adequam ao seu perfil (figura 4).

Para acesso ilimitado a todas as funcionalidades é necessário que o utilizador se encontre registado no portal público. A autenticação pode ser efectuada através da conta de acesso criada nesse portal ou através da conta de Facebook associada ao utilizador e registada na parte pública. O utilizador autenticado tem acesso às principais informações da sua conta. Através deste item o utilizador pode configurar o seu perfil, para que os dados manipulados estejam contextualizados com as suas preferências. Para além disso, através deste menu é possível aceder à lista das reservas efectuadas pelo utilizador e todos os dados referentes a estas (figura 5).

Na aplicação móvel desenvolvida, existe ainda uma ferramenta que permite a captura de imagens e posterior conversão do *QR Code*, para a respectiva acumulação de pontos. Nesta aplicação há ainda espaço para as funcionalidades ligadas às redes sociais, nomeadamente a implementação de mecanismos que permitem inserir comentários sobre pontos de interesse que estejam próximos da posição inicial do utilizador. Estes comentários são publicados no "Mural" da página do Facebook respeitante ao utilizador.

Figura 5. Exemplos de interface (2)

4.2 Desenvolvimento do sistema

A aplicação móvel foi desenvolvida para Android, sendo a base de dados local *SQLite*. Sempre que necessita aceder ou manipular informação externa acede a WEB Services desenvolvidos em *.Net*. A comunicação é efectuada através da biblioteca *KSOAP2* que permite consumir em *Java*, Web Services *.Net*. As interfaces da aplicação móvel foram criadas de acordo com as especificações para desenvolvimento de aplicações para Android, sendo a linguagem base *XML*.

A localização e actualização da posição geográfica de cada dispositivo é efectuada através do GPS do próprio aparelho, através da classe *LocationListener*, designadamente os métodos *onLocationChanged(Location loc)*, *onProviderDisabled(String provider)*, *onProviderEnabled(String provider)* e *onStatusChanged(String provider, int status, Bundle extras)*. No modo *online*, a visualização dos mapas é efectuada através da biblioteca externa do Google Maps, sendo para tal necessário estender a classe *MapActivity* e os métodos que possui. No modo *offline*, o mapa é desenhado através das facilidades oferecidas pela biblioteca *Nutiteq*, em que os diferentes *tiles* encontram-se armazenados localmente no dispositivo. Estes *tiles* são oferecidos pela *OpenStreetMap* e posteriormente é efectuado um pré-processamento, para delimitação das zonas e níveis de visualização. A posição actual do dispositivo móvel é desenhada sobre o mapa, através das facilidades oferecidas pela interface *MyLocationOverlay*. Os pontos de interesse são desenhados através da classe abstracta *ItemizedOverlay* que implementa um array de *OverlayItem*, possibilitando assim a criação de vários *markers*. Para implementação dos mecanismos de alerta foi necessário utilizar métodos oferecidos pela classe *LocationManager*, principalmente o *addProximityAlert(double latitude, double longitude , float radius, long expiration, PendingIntent intent)*. Este método é responsável por lançar a actividade, que faz surgir as notificações de pontos próximos. As notificações são efectuadas através de alertas visuais e sonoros.

O perfil do utilizador é definido através de filtros que operam sobre as diferentes categorias e do raio de alcance entre eventos próximos. Estes filtros para além de estarem associados à conta do utilizador, são também armazenados localmente e sempre que a aplicação é iniciada são carregados sem que seja necessária autenticação. Através da aplicação móvel cada utilizador pode configurar o seu perfil para que a toda informação recebida esteja de acordo com as suas preferências. Assim sendo, para o caso do Douro foram definidas 4 categorias principais de filtragem, respectivamente: *Turismo, Vinhos, Paisagem, Saberes e Sabores*. Cada uma destas categorias representa um conjunto de pontos de interesse de características similares e eventos associados. Existem ainda uma quinta categoria independente do caso de aplicação, que permite ao utilizador definir se quer explorar a região através de *QR Codes*. O utilizador para cada categoria pode definir qual a percentagem de resultados que pretender obter, sendo a ordem definida pela proximidade da posição espacial. Ou seja, se for definido 50% na categoria de Paisagem, o utilizador tem acesso a metade das subcategorias de informação que se encontram disponíveis nesta categoria (as mais relevantes), e que esta englobada no raio de alcance definido no mesmo formulário. As funciona-

lidades ligadas às redes sociais foram implementadas com a utilização dos mecanismos oferecidos pela biblioteca *fbconnect-android*, nomeadamente para acesso à conta do utilizador e respectiva informação mais relevante. A publicação de comentários acerca dos pontos de interesse georreferenciados também é possível através dos métodos desta biblioteca. Para a conversão dos *QR Codes* foi utilizada a API *KAYWA QR-Code*. Cada reserva do utilizador possui um código único e unipessoal, sendo constituído por 6 elementos alfanuméricos.

5 Conclusões e Trabalho Futuro

Após o trabalho efectuado, considera-se que a plataforma desenvolvida atinge os propósitos da criação de Novos Serviços para *Mobile Advertising*. Desenvolveu-se uma solução genérica, adaptável a qualquer região turística e que permite criar serviços de valor acrescentado. Estes serviços pouco intrusivos, são fornecidos de acordo com a localização do utilizador e tendo em conta o seu perfil de utilização.

Para que a metodologia funcione é necessário que na região exista uma estrutura organizacional semelhante à proposta, ou seja, que estejam bem definidos quais os pontos de interesse que pertencem a uma determinada organização. Para além disso, devem ser especificados quais os privilégios de cada organização e quais as competências e políticas da administração da plataforma, através da celebração de vários contractos. A aplicação só terá interesse para os utilizadores, caso exista uma boa cobertura documental e georreferenciada da região, logo será necessário estabelecimento de protocolos com entidades que possuem essas informações.

Relativamente ao trabalho futuro, a curto prazo, há a salientar o facto de que algumas das funcionalidades implementadas se encontrarem ainda numa fase de teste, sendo por isso necessário desenvolver padrões de testes que contemple a avaliação da solução a nível de performance, sustentabilidade de carga e interacção com os utilizadores. É também necessário, criar mecanismos de optimização no que diz respeito à actualização da informação relativa a pontos de interesse que sejam fornecidos por entidades externas, para servirem de complemento aos dados existentes. A médio prazo, os objectivos passarão pela implementação de mecanismos relacionados com a realidade aumentada, podendo substituir a tecnologia *QR Code* ou conceber novos serviços que criem mais valor acrescentado. Para ser melhorada a acessibilidade da aplicação móvel, a longo prazo podem ser implementadas funcionalidades para reconhecimento e síntese de voz, permitindo por exemplo, a invisuais mais igualdade de acesso. A utilização da recente tecnologia Google Goggles também pode vir a ser uma mais-valia, na criação de novos serviços para *Mobile Advertising*.

Referências

1.World Travel & Tourism Council, The Economic Impact of Trave & Tourism Portugal 2010, Disponível em `http://www.wttc.org/download.php?file=http://www.wttc.org/bin/pdf/original_pdf_file/portugal.pdf`

2.Ministério da Economia e da Inovação, Plano Estratégico Nacional do Turismo 2006-2015, Disponível em `http://www.portugal.gov.pt/pt/Documentos/Governo/MEI/PENT.pdf`

3.Mobile Marketing Association, Mobile Advertising Guidelines, Disponível em `www.mmaglobal.com/mobileadvertising.pdf`

4.Afonso da Fonte Gomes Vaz, Fornecimento de Serviços Push Direccionados e Baseados em Localização, tese de mestrado em Engenharia de Redes de Comunicações, Universidade Técnica de Lisboa, 2009.

5.André Filipe Ferreira Cardoso, Publicidade móvel adaptada ao utilizador, tese de mestrado em Engenharia Informática - ramo Computação Móvel, Universidade Fernando Pessoa, 2008.

6.Hugo Miguel Meireles Teixeira, Aplicação Móvel com Localização Geográfica, tese de mestrado em Engenharia de Redes de Comunicações, Universidade Técnica de Lisboa, 2009.

7.Timo Ojala. Case studies on context-aware mobile multimedia services. Journal on Digital Information Management 8(1):3-14. 2010.

8.Hinze, A., Malik, P., and Malik, R. Interaction design for a mobile context-aware system using discrete event modelling. In Proceedings of the 29th Australasian Computer Science Conference - Volume 48 (Hobart, Australia, January 16 - 19, 2006). V. Estivill-Castro and G. Dobbie, Eds. ACM International Conference Proceeding Series, vol. 171. Australian Computer Society, Darlinghurst, Australia, 257-266. 2006.

9.Mark Amundson, M. Compass Assisted GPS for LBS Applications, Honeywell, http://www.honeywell.com/magneticsensors, 2006.

10.P. Pombinho, A. P. Afonso, M. B. Carmo, Contextos e Visualização Adaptativa em Ambientes Móveis. 1º INForum - Simpósio de Informática, Setembro, 2009.

11.S. Freitas, M. Silva e A.P. Afonso, Geotumba móvel: motor de busca geográfico para dispositivos móveis, Encontro Nacional de Visualização Científica (ENVC 05), Setembro, 2005.

12.Pelosi, G. and Psaila, G. 2010. SMaC: spatial map caching technique for mobile devices. Em: Proceedings of the 2010 ACM Symposium on Applied Computing, SAC '10, 2010.

13.A. Vaz, P. Pombinho, A. P. Afonso, M. B. Carmo, MoViSys - A Visualization System for Mobile Devices. Em: Springer-Verlag (Ed.), Visual 2008 - 10th International Conference on Visual Information Systems, LNCS 5188, p. 167-178, Setembro, 2008.

14.P. Pombinho, M.B. Carmo e A.P. Afonso, Visualização de informação georeferenciada em dispositivos móveis, V Encontro Português de Computação Gráfica (EPCG 07), Outubro 2007.

15.K. Church, J. Neumann, M. Cherubini, N. Oliver, The "Map Trap"?: an evaluation of map versus text-based interfaces for location-based mobile search services. Em WWW '10: Proceedings of the 19th international conference on World wide web, ACM, p. 261–270, 2010.

16.Hinze, A. and Junmanee, S. "Travel recommendations in a mobile tourist information system." Kaschek, R., Mayr, H. C. and Liddle, S. (eds), Proc Fourth International Conference on Information Systems Technology and its Applications (ISTA 2005), Palmerston North, New Zealand, 86-100. Gesellschaft für Informatik, Bonn. 2005.

Um Sistema Publicador/subscritor com Subscrições Geograficamente Distribuídas para RSSFs *

Ricardo Mascarenhas e Hugo Miranda

Universidade de Lisboa
Faculdade de Ciências
LaSIGE
{rmascarenhas@lasige.di.fc.ul.pt,hmiranda@di.fc.ul.pt}

Resumo As redes de sensores sem fios (RSSFs) são compostas por um grande número de pequenos dispositivos com restrições a nível energético, capacidade de processamento e de memória que monitorizam o ambiente em que estão inseridas. As fortes restrições a que estão sujeitas obrigam à utilização de paradigmas de comunicação específicos, que tenham em consideração uma utilização racional dos recursos.
Este artigo apresenta um sistema Publicador/Subscritor que tem em conta o interesse dos nós em determinados tipos de informação e suprime o envio de informação sem relevância. Para além disso, o sistema adapta-se às restrições de energia e de memória dos dispositivos.

1 Introdução

Com a evolução da tecnologia e consequente diminuição do tamanho dos dispositivos, as redes de sensores sem fios (RSSFs) tornaram-se extremamente apelativas. As RSSFs não necessitam de uma infra-estrutura de suporte à comunicação, e são tipicamente compostas por dispositivos de pequena dimensão com baixo poder computacional, memória e energia limitada. Os sensores estão equipados com tecnologia que lhes permite obter informação sobre o ambiente em que estão inseridos (por exemplo: temperatura, humidade, pressão, movimento) e estão tipicamente dispersos pela área de observação onde recolhem dados e, após algum processamento, os enviam para um ou mais nós *sink*, que se encarregam de fazer chegar a informação ao utilizador final. A utilização deste tipo de redes é apropriada para um vasto número de aplicações, como por exemplo, monitorização de habitats, agricultura de precisão, monitorização de edifícios e aplicações militares [1, 7, 18].

Os sensores estão equipados com uma interface de rede sem fios que lhes permite enviar informação directamente para os nós que se encontrem dentro do seu raio de transmissão. A entrega de mensagens a nós mais distantes é

* This work has been partially supported by FCT project REDICO (PTDC/EIA/71752/2006) through POSI and FEDER.

conseguida por retransmissões sucessivas, realizadas por diferentes dispositivos. A descoberta de uma sequência de nós entre a origem e o destino da mensagem é realizada por protocolos de encaminhamento para redes não infra-estruturadas.

Tipicamente, os nós sensores são programados em tempo de produção para utilizar a sua limitada capacidade de processamento na avaliação da relevância da informação obtida, ou seja, se a sua retransmissão justifica o consumo de energia necessário para a entrega ao *sink* mais próximo. Com o aumento do número de sensores, aumenta também o volume e diversidade de informação recolhida. Adicionalmente, abre-se caminho à especialização dos nós *sink*, que poderão seleccionar, de forma independente dos restantes, os tipos de informação que consideram relevantes. Neste cenário os sensores passam a ter que identificar correctamente o(s) *sink(s)* a quem a informação deve ser entregue. Impõe-se por isso a definição de novos modelos de comunicação, que contemplem adequadamente casos em que nenhum nó *sink* esteja interessado em algumas das amostras, ou esteja interessado apenas quando um conjunto de factores seja satisfeito. A incapacidade de incorporar condições como as anteriores no modelo de comunicação pode levar ao envio desnecessário de um elevado número de mensagens, que implicam um gasto energético não negligenciável e consequente diminuição de longevidade da rede [9].

Os sistemas Publicador/Subscritor [10] (Pub/Sub) fornecem as funcionalidades necessárias para o modelo dinâmico de RSSFs apresentado acima. Num sistema Pub/Sub os *publicadores* produzem *publicações*, unidades de informação não endereçadas que são entregues ao sistema. Os *subscritores*, por sua vez, expressam perante o sistema o seu interesse em publicações com determinadas características. Cabe ao sistema Pub/Sub a responsabilidade de entregar as publicações aos subscritores que nelas estejam interessados. Este modelo de comunicação molda-se perfeitamente às necessidades das RSSFs descritas em cima pois a notificação da produção de informação de interesse contribui para a redução do número de mensagens desnecessárias.

As concretizações descentralizadas de sistemas Pub/Sub para redes sem fios poderão ser posicionadas entre dois extremos. Um consiste na inundação das subscrições, onde todos os nós retransmitem as subscrições escutadas. Deste modo, as subscrições chegam a todos nós da rede permitindo que os publicadores possam determinar localmente quais os subscritores a quem a publicação deverá ser entregue. Este extremo terá maior eficiência num cenário onde importa optimizar o custo da entrega de publicações, ou seja, quando se espera que o número de publicações exceda o número de subscrições. Redes de monitorização de longa duração, com uma filiação aproximadamente constante ao longo do tempo são um bom exemplo deste tipo de aplicação. O outro consiste na inundação das publicações, delegando nos nós a responsabilidade de transmitir e verificar se a publicação satisfaz a sua subscrição. Este extremo será preferível para redes onde se espera um número muito reduzido de publicações, por exemplo aquelas cujo objectivo é a notificação de um único evento, como um fogo florestal. Contudo, nenhum dos extremos tem uma concretização adequada para RSSFs. Uma vez que cada transmissão/recepção efectuada tem um custo energético não neg-

ligenciável [9], importa encontrar soluções que reduzam o número de operações de transmissão a um mínimo. Do mesmo modo, a inundação de subscrições não é escalável por poder vir a consumir uma larga porção da escassa memória dos sensores.

Neste artigo é proposta e avaliada uma solução de compromisso, em que a informação relativa às subscrições é parcialmente replicada pelos dispositivos que compõem a rede. As subscrições são distribuídas de modo a que um qualquer publicador as possa obter, independentemente da sua localização, com um baixo custo. As publicações são enviadas utilizando uma versão de um protocolo de encaminhamento convencional para redes não infra-estruturadas adaptado aos objectivos do trabalho. O remanescente do artigo está organizado da forma descrita em seguida. A Sec. 2 apresenta em mais detalhe o modelo Publicador/Subscritor e alguns dos trabalhos realizados no âmbito das redes sem fios. O sistema proposto é apresentado na Sec. 3 e avaliado na Sec. 4. Finalmente, a Sec. 5 conclui o artigo e discute o trabalho futuro.

2 Trabalho relacionado

Uma das características distintivas dos diferentes sistemas de Pub/Sub existentes é a expressividade com que um subscritor consegue indicar os seus interesses. No modelo baseado em tópicos (por exemplo, [20,22]), quer as publicações, quer as subscrições são identificadas por um assunto. Os subscritores são notificados sempre que o assunto da informação publicada satisfaça o assunto especificado nas subscrições. Este modelo tem como principal desvantagem a fraca expressividade que oferece aos subscritores contudo, é facilmente projectado num modelo de difusão selectiva (*multicast*), com cada tópico a corresponder a um endereço distinto.

O modelo baseado em conteúdos oferece uma maior expressividade já que permite a utilização de operadores sobre os atributos das subscrições e não apenas sobre o tópico. Neste modelo assume-se que um subscritor está interessado numa publicação se e só se todas as restrições declaradas na subscrição são satisfeitas. Este modelo é indicado para RSSF em que os nós *sink* podem refinar os seus interesses de acordo com as suas necessidades. Por exemplo, uma aplicação para detecção de incêndios estaria exclusivamente interessada em publicações registando temperatura superior a 60° e humidade inferior a 70%. Alguns sistemas que usam este modelo estão descritos em [3,6,8,15,17].

Existem diversas estratégias de concretização de sistemas Pub/Sub [10]. Alguns sistemas centralizam a correspondência entre as publicações e os subscritores em nós bem conhecidos, denominados mediadores (*brokers*). Apesar de simplificarem consideravelmente a complexidade do sistema, a forte dependência num conjunto limitado de dispositivos, e os recursos computacionais requeridos tornam os brokers inadequados para os cenários de aplicação das RSSF.

Os sistemas descentralizados não recorrem a *brokers*, distribuindo a responsabilidade pelos participantes pelo que são o mais adequado para RSSFs. A descentralização do sistema Pub/Sub é uma mais valia mas apresenta um con-

junto de desafios acrescidos que resulta do aumento da complexidade resultante da distribuição. No caso das RSSFs estamos interessados em soluções que requeiram simultaneamente baixa utilização da memória e a transmissão de um número limitado de mensagens por todos os participantes.

O SPINE [4] é um sistema de Pub/Sub para redes parcialmente infra-estruturadas (*mesh*). O sistema aproveita o facto de a rede dispor de alguns dispositivos sem restrições de energia e com uma localização fixa, os quais designa de *routers*, utilizando-os para armazenar as subscrições. O SPINE assume que os *routers* estão dispostos em matriz. As subscrições são disseminadas pelos routers que se encontrem na mesma linha que o subscritor e as publicações pelos routers que se encontram na mesma coluna. Sempre que os routers recebem uma publicação, verificam se há subscritores interessados na sua linha e encaminham-na para o subscritor. Apesar da sua simplicidade, a aplicação deste sistema a RSSFs não é trivial já que obrigaria a que os nós sensores tivessem o conhecimento da sua localização geográfica.

O objectivo do Data Centric Storage (DCS) [19] é distribuir e localizar facilmente informação numa rede não infra-estruturada. Os dados são identificados por uma chave e a esta é associado um par de coordenadas geográficas através de uma função determinista. Os participantes responsáveis por salvaguardar cada item de dados são aqueles que formam um círculo em torno da coordenada geográfica determinada para o item. Uma aplicação possível do DCS é atribuir uma chave a cada tópico de um sistema Pub/Sub, nomeando assim os dispositivos mais próximos da localização como mediadores [2]. Contudo, este modelo não é adequado para sistemas Pub/Sub baseados em conteúdos. Adicionalmente, o DCS requer que todos os nós tenham conhecimento das suas coordenadas geográficas e dos limites da área de rede, o que diminui a aplicabilidade do sistema.

3 Sistema Pub/Sub

Para colmatar as limitações impostas pelos dispositivos que compõem uma RSSFs, propomos um sistema Pub/Sub descentralizado baseado em conteúdos. A nossa aproximação assenta numa replicação parcial das subscrições mas com as réplicas distribuídas geograficamente. Enquanto a replicação parcial diminui a quantidade de memória utilizada em cada dispositivo, a distribuição geográfica das réplicas contribui para um baixo consumo energético, uma vez que o conjunto integral das subscrições pode ser obtido dos vizinhos de qualquer nó da rede. No nosso sistema, cabe aos publicadores recolher as subscrições e entregar as publicações aos subscritores, utilizando encaminhamento ponto-a-ponto.

A Fig. 1 ilustra a arquitectura do sistema proposto. A interacção entre a aplicação e o sistema de Pub/Sub é efectuada através das primitivas *Subscrever* e *Publicar*. A primitiva *Notificação* dá a conhecer à aplicação as publicações que satisfazem a subscrição realizada. O nível Pub/Sub utiliza os serviços de dois módulos: *i*) um serviço de replicação parcial com distribuição geográfica para redes não infra-estruturadas denominado PCACHE [14], encarregue da replicação e recolecção das subscrições; e *ii*) o protocolo de encaminhamento

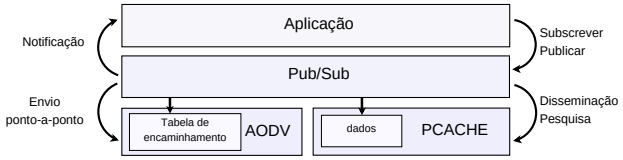


Figura 1. Arquitectura do sistema

AODV [16] através do qual são disseminadas as publicações. Cada um destes módulos disponibiliza ao módulo de Pub/Sub acesso para leitura de uma estrutura de dados. A tabela de encaminhamento do AODV é utilizada pelo módulo de Pub/Sub para identificar troços comuns nas rotas para os subscritores. A PCACHE mantém na *zona de dados* a informação que coube ao dispositivo armazenar durante a execução do algoritmo de replicação. No contexto deste trabalho a zona de dados mantém por isso um subconjunto das subscrições efectuadas. Esta é acedida pelo módulo de Pub/Sub para identificação da lista parcial de subscrições que satisfazem uma publicação.

3.1 Distribuição geográfica da informação

A PCACHE [14] é um módulo de código intermédio para redes sem fios não estruturadas que replica os dados de tal forma que as réplicas estejam suficientemente distantes para prevenir o excesso de redundância, mas simultaneamente permanecem suficientemente perto de forma a que a informação possa ser obtida por qualquer nó com um número de mensagens reduzido. Esta secção descreve de forma muito abreviada a PCACHE, salientando os aspectos mais relevantes para este trabalho. Informação adicional poderá ser encontrada em [11, 12, 14].

A PCACHE não faz qualquer interpretação sobre os tipos de dados armazenados, assumindo na sua versão mais simples que estes são constituídos por um par <chave,valor>, sendo a chave utilizada para a identificação de duplicados e como critério de pesquisa. É assumido que os dispositivos disponibilizam memória para que a PCACHE possa operar embora não seja necessário que todos os dispositivos disponibilizem a mesma quantidade de memória, facto que possibilita a utilização de diferentes tipos de sensores na mesma rede. O funcionamento da PCACHE não se baseia em informação de localização como o GPS, requerendo apenas que os dispositivos tenham a capacidade de obter a força de sinal (RSSI – Received Signal Strength Indication) das mensagens recebidas.

3.2 Subscrições

As subscrições são disseminadas utilizando o mecanismo base da PCACHE, assegurando por isso a distribuição geográfica das réplicas. A informação associada a cada subscrição e salvaguardada na PCACHE é constituída por um filtro de pesquisa e pelo endereço do subscritor. Quando comparado com uma publicação,

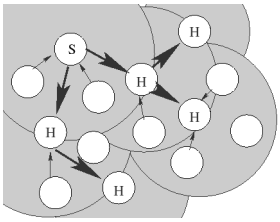


Figura 2. Recolha de subscrições

o filtro de pesquisa permite identificar se a publicação satisfaz ou não os critérios de subscrição de acordo com o modelo baseado em conteúdos.

Na PCACHE a disseminação de dados é realizada por difusão, utilizando o algoritmo PAMPA [13]. A decisão de armazenar um determinado item de dados é local a cada nó mas condicionado pelo conteúdo das mensagens transmitidas pelos vizinhos e pela memória disponível.

3.3 Publicações

A operação de disseminação é decomposta em duas fases: identificação das subscrições relevantes e entrega das publicações. Ambas são geridas pelo publicador.

Recolha de Subscrições A recolha de subscrições tem como objectivo reunir no publicador os endereços dos subscritores a quem será destinada a publicação. Para tal, é utilizada a operação de pesquisa da PCACHE, passando o publicador a própria publicação como critério de pesquisa. Os mecanismos de difusão e pesquisa da PCACHE são ortogonais ao tipo de dados utilizado. Por essa razão, a operação de verificação de correspondência, entre os dados guardados em cada nó pela PCACHE e a publicação, é delegada pela PCACHE na instância local do módulo de Pub/Sub. Contudo, cabe à PCACHE o envio das respostas, que respeita o algoritmo de consulta original. Este algoritmo assegura um consumo mínimo de energia, por exemplo por agregar as respostas de diferentes participantes nos nós intermédios e removendo duplicados. O algoritmo de recolha está representado na Fig. 2. Na figura, a recolha é iniciada pelo nó S e propagada pelos nós assinalados com H. Ao receber a primeira cópia da mensagem, cada nó verifica a sua zona de dados local e caso disponha de subscrições que satisfaçam a publicação envia a informação disponível para o nó H, o qual procede à agregação dos dados e os envia por sua vez para o nó anterior.

Envio das publicações A PCACHE fornece um serviço de pesquisa em melhor esforço. Em particular, em nenhum instante é possível assegurar ao participante que realiza a operação de recolha de subscrições que *i*) todos os itens de dados que satisfazem a pesquisa foram retornados; *ii*) que não serão recebidas mais respostas. Contudo, conhecendo o algoritmo de pesquisa da PCACHE, é possível determinar um período ao fim do qual todas as respostas já deverão ter sido

entregues e que é proporcional ao número de saltos realizado pela pesquisa. Durante o tempo de espera, o publicador concentra as respostas recebidas numa fila, removendo eventuais duplicados que resultarão de duplicação de registos em nós distintos da rede. Expirado o temporizador no publicador, este tem em principio toda a informação sobre eventuais subscritores, isto é, o endereço dos nós interessados na publicação. As publicações podem agora ser enviadas em mensagens ponto-a-ponto, recorrendo ao protocolo de encaminhamento para redes não infra-estruturadas AODV [16], depois de aplicada uma optimização que reduz o número de mensagens necessárias para a entrega das publicações.

O AODV é um protocolo de encaminhamento reactivo, ou seja, que procura rotas para cada destino apenas quando necessário. A tabela de encaminhamento mantida em cada nó tem uma entrada por destinatário onde, para além de outra informação de controlo, como a actualidade da rota, consta o endereço do próximo nó da rota para o destinatário. O módulo de Pub/Sub utiliza a tabela de encaminhamento do AODV para agregar os destinos de acordo com o próximo nó da rota para o destino. Para cada nó seguinte é enviada uma única cópia da publicação, juntamente com a lista de destinatários a quem a publicação deverá ser entregue. De notar que a mensagem é endereçada ao nó seguinte e não a um dos destinatários pelo que é recebida pelo módulo de Pub/Sub deste nó. Cada vez que um nó recebe a publicação contendo vários destinatários consulta a sua tabela de encaminhamento e repete o processo, enviando tantas mensagens quantos nós seguintes nas rotas para os destinatários. A aplicação da optimização termina quando, no publicador, ou num qualquer nó intermédio, o nó seguinte não é partilhado por mais nenhum destinatário, sendo então a entrega delegada no AODV, através de uma mensagem endereçada ao destinatário.

As respostas entregues tardiamente pela PCACHE não beneficiam do agrupamento. A optimização também não é aplicada aos destinatários para os quais não exista uma entrada na tabela de rotas do publicador. Nestes casos, é criada uma mensagem endereçada ao destinatário e o envio é delegado no AODV. Na presença de um elevado número de operações de descoberta de rota num curto intervalo de tempo, pode surgir instabilidade na rede, um cenário vulgarmente denominado por *Broadcast Storm* [21]. Para reduzir o risco da sua ocorrência, as mensagens com destino a nós que irão provocar operações de pedido de rota são entregues ao módulo AODV a um ritmo cadenciado, permitindo a estabilização da rede entre cada uma das operações.

4 Avaliação

O desempenho da solução proposta é comparado com uma versão melhorada da inundação de publicações por utilizar o algoritmo de difusão PAMPA. As aproximações por inundação de publicações têm custo nulo na difusão das subscrições mas um custo que se espera mais elevado por publicação. Importa por isso perceber o ponto a partir do qual o equilíbrio da nossa solução apresenta um custo inferior ao da inundação, sabendo que a solução apresentada terá um custo superior (em número de transmissões) à inundação de subscrições. No entanto,

importa referir que a inundação de subscrições troca a replicação parcial pela replicação total das subscrições, o que representa um aumento do consumo de memória que pode não ser comportável face aos limitados recursos dos dispositivos.

A avaliação é realizada utilizando um protótipo do sistema Pub/Sub estudado desenvolvido para a v. 2.32 do simulador de redes *ns-2*[1]. O mesmo protótipo é utilizado para avaliar o desempenho da inundação de publicações, através da contabilização do número médio de retransmissões realizadas pelo algoritmo PAMPA para a disseminação das subscrições. O protótipo utiliza a concretização do AODV que acompanha o simulador, tendo sido apenas desactivado o temporizador de invalidação de rotas da tabela de encaminhamento por não apresentar qualquer vantagem para cenários sem movimento. Para não prejudicar a avaliação da qualidade de distribuição em condições ideais, o protótipo assume que os dispositivos têm memória suficiente para guardar todas as subscrições que a PCACHE determina, sabendo-se que serão sempre, para cada dispositivo, uma pequena fracção do total [14].

O cenário de simulação é composto por 100 nós uniformemente dispostos por regiões com 8 dimensões distintas, definindo desta forma redes com diferentes densidades. Cada participante dispõe de uma interface de rede sem fios IEEE 802.11b a 11Mb/s, com um raio de transmissão fixo de 250 metros. Desta forma, as diferentes dimensões da rede simulada apresentam também comprimentos de rotas distintos. Os resultados apresentados resultam da execução de 20 simulações para cada área de simulação. Cada simulação combina diferentes disposições dos nós na área de simulação, diferentes escolhas dos atributos nas subscrições e nas publicações, e diferentes momentos de subscrição e publicação.

Cada simulação tem a duração de 87120 segundos e é decomposta em 2 fases. Nos primeiros 660 segundos, 50 nós realizam outras tantas subscrições. O momento de cada subscrição é seleccionado aleatoriamente, reservando-se os últimos 60s para a terminação das operações de subscrição. A partir dos 660s, são efectuadas 1440 publicações a uma média de uma publicação por minuto. Uma vez mais não são iniciadas publicações nos últimos 60s, assegurando desta forma o tempo necessário para a entrega das publicações a todos os subscritores.

Para simular um modelo baseado em conteúdos, cada publicação é representada por um tuplo de 4 atributos da forma $< x_1, x_2, x_3, x_4 >, x_i \in [1, 9]$. Os valores de cada atributo são independentes, gerados aleatoriamente com uma distribuição uniforme. As subscrições por sua vez impõem restrições aos 4 atributos, na forma $< [y_1, y_1 + 3], [y_2, y_2 + 3], [y_3, y_3 + 3], [y_4, y_4 + 3] >, y_i \in [1, 6]$, com y_i a ser determinado aleatoriamente de acordo com uma distribuição zipf. Esta distribuição foi escolhida por ter sido demonstrado que representa por exemplo, índices de popularidade de sítios web [5] ou frequências de utilização de palavras, cenários que se aproximam das aplicações antecipadas para os sistemas Pub/Sub. A combinação da distribuição uniforme das publicações com a distribuição zipf das subscrições é ilustrada na Fig. 3, onde para cada uma das

[1] `http://www.isi.edu/nsnam/ns/`

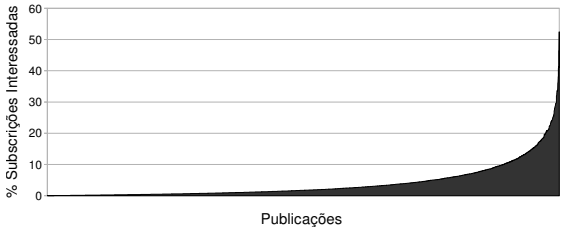


Figura 3. Interesse das subscrições face às publicações produzidas

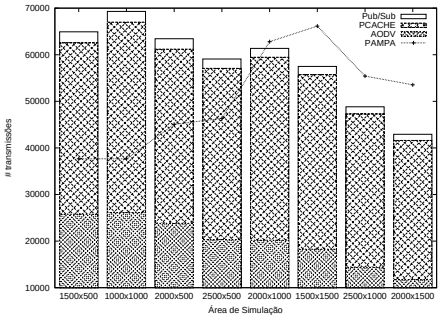


Figura 4. Número total de transmissões

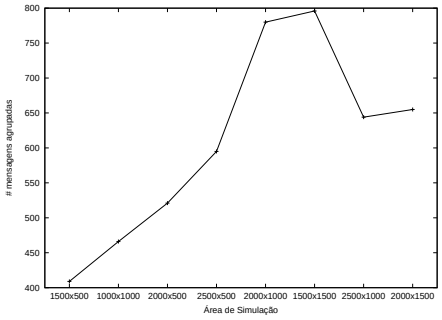


Figura 5. Reutilização das rotas

6561 (9^4) publicações possíveis é apresentada a proporção de 10000 subscrições que satisfaz.

A Fig. 4 mostra o número de mensagens enviadas em função da área de simulação. Para o sistema Pub/Sub, é ainda representada a contribuição dos diferentes protocolos utilizados para o número total de transmissões. A figura mostra que o número total de transmissões das duas soluções avaliadas evolui em sentidos opostos com a densidade. As transmissões PAMPA tendem a aumentar com a diminuição da densidade, o que se deve ao facto de o PAMPA ter de adaptar a proporção de nós que retransmitem as mensagens à densidade da rede [13]. O PAMPA também é utilizado no sistema Pub/Sub, nomeadamente, nas operações de disseminação e recolha de subscrições. Contudo, uma vez que a recolha de subscrições consiste numa difusão limitada a alguns saltos, o aumento do número de retransmissões com a densidade é atenuado pelos ganhos ao nível do AODV. Enquanto a solução de inundação de publicações apresenta um custo (em número de mensagens) fixo por publicação para cada densidade, o sistema proposto apresenta um custo que reduz com o aumento do número de publicações, uma vez que o número de operações de descoberta de rotas é progressivamente menor. Este facto justifica o maior número de mensagens do sistema proposto em comparação com a inundação de publicações nas densidades mais elevadas. A redução do número de mensagens observada nos testes realizados nas duas menores densidades é atribuído a partições de rede pelo que os resultados não podem ser considerados.

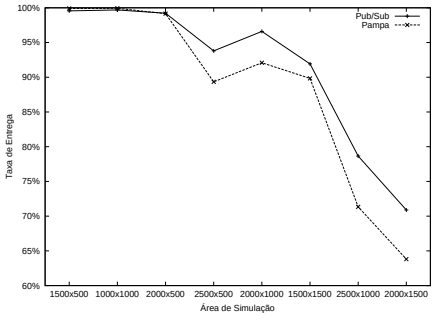


Figura 6. Taxa de entrega

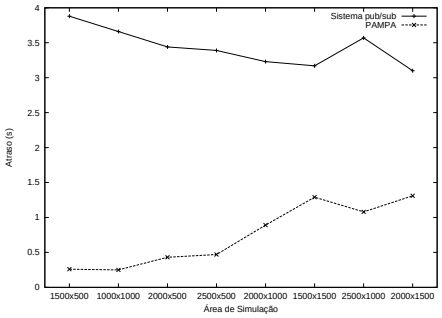


Figura 7. Latência

A redução do número de transmissões AODV é explicada pela Fig. 5, que contabiliza o número de cópias de publicações agrupadas pelo publicador. Como é ilustrado na figura, a diminuição da densidade da rede é acompanhada de um aumento do número de mensagens agrupadas. Com o aumento da distância entre os nós, o número de nós que o AODV pode seleccionar para a construção de rotas diminui. Por esta razão o número de rotas possíveis também diminui, aumentando a probabilidade das mensagens partilharem o mesmo nó seguinte, o que resulta num aumento do número de mensagens agrupadas.

A Fig. 6 apresenta a taxa de entrega de publicações e subscrições. A figura mostra que na ausência de partições a taxa de entrega se mantém acima dos 90%. É perceptível o padrão referente ao decréscimo da taxa de entrega das publicações à medida que a taxa de entrega do PAMPA também decai. A quebra progressiva da taxa de entrega com a diminuição da densidade é atribuída ao isolamento de alguns dispositivos que são por isso incapazes de anunciar as suas subscrições, o que resulta num conhecimento insuficiente dos subscritores e por consequência na descida da taxa de entrega do sistema Pub/Sub.

O processo de recolha de subscrições levado a cabo pelo sistema Pub/Sub introduz naturalmente algum atraso que não é possível colmatar. Os resultados apresentados para a latência da disseminação na Fig. 7, confirmam estas expectativas. Os valores para o sistema Pub/Sub medem o tempo médio decorrido entre a produção da publicação e a sua entrega a cada subscritor. No caso do PAMPA, mede-se o tempo entre a produção da publicação e a sua entrega ao último nó da rede. O gráfico sugere a existência de uma tendência de convergência entre as duas aproximações avaliadas. No caso do sistema Pub/Sub, a diminuição é atribuída à reutilização de rotas, dado que cada transmissão sofre intencionalmente um atraso para assegurar o envio cadenciado de mensagens que evite perturbações na rede. No caso do PAMPA, o crescimento resulta do aumento do número de transmissões e do algoritmo executado pelo protocolo, uma vez que cada dispositivo aplica algum atraso à sua própria retransmissão [13].

5 Conclusões e Trabalho Futuro

Este artigo apresenta um sistema Pub/Sub completamente descentralizado que se adapta às restrições e características das RSSFs. A solução apresentada tem como objectivo ser conservadora no uso de recursos, em particular, nos recursos energéticos e na memória usada para armazenamento das subscrições.

A distribuição geográfica das subscrições é efectuada pelos subscritores através da plataforma de código intermédio PCACHE, que apresenta duas vantagens: para além de requerer um número reduzido de transmissões, faz uma utilização racional da escassa memória disponível nos sensores. Os publicadores independentemente da sua localização reúnem informação sobre as subscrições, e enviam as publicações aos respectivos subscritores.

A redução do consumo energético é alcançado através da redução do número de mensagens, conseguindo ainda assim uma taxa de entrega relativamente alta. A latência do sistema sugere que este não é adequado para aplicações com requisitos temporais.

Para trabalho futuro, planeamos variar alguns parâmetros e desenvolver outras optimizações, nomeadamente, estratégias para diminuição da latência do sistema. O estudo do impacto de outros protocolos de encaminhamento e da exploração de outras sinergias com o AODV fazem também parte dos planos para futuros melhoramentos.

Referências

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Computer Networks 38(4), 393 – 422 (2002)
2. Albano, M., Chessa, S.: Publish/subscribe in wireless sensor networks based on data centric storage. In: CAMS '09: Proceedings of the 1st International Workshop on Context-Aware Middleware and Services. pp. 37–42 (2009)
3. Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R., Sturman, D.: An efficient multicast protocol for content-based publish-subscribe systems. In: Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on. pp. 262–272 (1999)
4. Briones, J.A., Koldehofe, B., Rothermel, K.: SPINE: Publish/Subscribe for Wireless Mesh Networks through Self-Managed Intersecting Paths. In: Proceedings of the 8th International Conference on Innovative Internet Community Systems (I2CS 2008). Schoelcher, Martinique (2008)
5. Cao, L.B.P.: Web caching and zipf-like distributions: Evidence and implications. In: In INFOCOM. pp. 126–134 (1999)
6. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Trans. Comput. Syst. 19(3), 332–383 (2001)
7. Cerpa, A., Elson, J., Estrin, D., Girod, L., Hamilton, M., Zhao, J.: Habitat monitoring: application driver for wireless communications technology. SIGCOMM Comput. Commun. Rev. 31(2 supplement), 20–41 (2001)
8. Cugola, G., Di Nitto, E., Fuggetta, A.: Exploiting an event-based infrastructure to develop complex distributed systems. In: ICSE '98: Proceedings of the 20th international conference on Software engineering. pp. 261–270 (1998)

9. Feeney, L., Nilsson, M.: Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. Proc. of the 20th Conf. of the IEEE Computer and Communications Societies (INFOCOM 2001). Proceedings. pp. 1548–1557 (2001)
10. Garbinato, B., Miranda, H., Rodrigues, L.: Middleware for Network Eccentric and Mobile Applications. Springer (2009)
11. Leggio, S., Miranda, H., Raatikainen, K., Rodrigues, L.: Sipcache: A distributed sip location service for mobile ad-hoc networks. In: Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference on (2006)
12. Miranda, H.: Gossip-Based Data Distribution in Mobile Ad Hoc Networks. Ph.D. thesis, Universidade de Lisboa, Campo Grande, 1749-016 Lisboa - Portugal (2007)
13. Miranda, H., Leggio, S., Rodrigues, L., Raatikainen, K.: A power-aware broadcasting algorithm. Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium. Proceedings. (2006)
14. Miranda, H., Leggio, S., Rodrigues, L., Raatikainen, K.: An algorithm for dissemination and retrieval of information in wireless ad hoc networks. In: Kermarrec, A.M., Bougé, L., Priol, T. (eds.) Proceedings of the 13th International Euro-Par Conference, Euro-Par 2007. Lecture Notes in Computer Science, vol. 4641, pp. 891–900. Springer (2007)
15. Pereira, J., Fabret, F., Llirbat, F., Preotiuc-Pietro, R., Ross, K.A., Shasha, D.: Publish/subscribe on the web at extreme speed. In: VLDB. pp. 627–630 (2000)
16. Perkins, C.E., Royer, E.M.: Ad-hoc on-demand distance vector routing. Mobile Computing Systems and Applications, IEEE Workshop on 0, 90 (1999)
17. Pietzuch, P.R., Bacon, J.M.: Hermes: A distributed event-based middleware architecture. Distributed Computing Systems Workshops, International Conference on (2002)
18. Pottie, G.J., Kaiser, W.J.: Wireless integrated network sensors. Commun. ACM 43(5), 51–58 (2000)
19. Ratnasamy, S., Karp, B., Shenker, S., Estrin, D., Govindan, R., Yin, L., Yu, F.: Data-centric storage in sensornets with ght, a geographic hash table. Mob. Netw. Appl. 8(4), 427–442 (2003)
20. Rowstron, A., Kermarrec, A.M., Castro, M., Druschel, P.: Scribe: The design of a large-scale event notification infrastructure. In: Lecture Notes in Computer Science. pp. 30–43. Springer (2001)
21. Tseng, Y., Ni, S., Chen, Y., Sheu, J.: The broadcast storm problem in a mobile ad hoc network. Wirel. Netw. 8(2/3), 153–167 (2002)
22. Zhuang, S., Zhao, B., Joseph, A., Katz, R., Kubiatowicz, J.: Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In: NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video. pp. 11–20 (2001)

Bluetooth Hotspots for Smart Spaces Interaction

Miguel M. Almeida, Helena Rodrigues, and Rui José

Departamento de Sistemas de Informação
Universidade do Minho

Abstract. We may find in the market many smart space applications and projects which are using Bluetooth services as mechanism for user-interaction. However, their ad-hoc implementation come as a limitation to their deployment in a global interactive model. This survey focuses on the requirements of such applications, systematizing their interaction models and presents the main key design of a middleware component for Bluetooth interaction-based pervasive applications, running on Linux routers.

1 Introduction

Bluetooth is present in most of the actual handheld devices. Being easy to use, a short-range and low cost technology, many applications are using it to interact with users. Despite many needs of these projects being similar, each one implements its own Bluetooth interface, deployed only for that purpose. So, why not have a shared component, capable of serving multiple applications simultaneously? This could empower and encourage the development of new applications based on the same situated interactions, freeing the developer from Bluetooth technical details and enabling a more sustainable development of the desired applications. The objective of this study is to justify the need of such component - a software running on Linux-enabled routers, offering Bluetooth services to applications. In this paper we survey a set of Bluetooth interaction-based projects leading to a set of reference scenarios, which we use as a starting point for the discussion of the main key design issues of a Bluetooth Hotspot, a middleware component for Bluetooth interaction-based pervasive applications.

2 Related Work

Our analysis of the related work is based on the study of representative systems based on most common Bluetooth services: device and service discovery, Bluetooth OBEX and RFCOMM connection. On the whole set, we mainly identify three different ways

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 439–442

to share information between the system and its users: 1) using the device name, 2) sending or receiving a file, or 3) establishing a permanent connection (e.g. socket) to share more complex data.

The simplest form of interacting with an application is using Bluetooth's device name following a predefined syntax. With this method, the user does not need to install a specific application, promoting a cross-platform method. The BluetunA project [1] proposes a system for people to share their music interests using the Bluetooth name on their devices. Users on public spaces may suggest music and artists to be played just changing their device names to specific tags. Bluetooth device scanning based systems also include car traffic monitoring. For example, in [2], Bluetooth scanners, geographically distributed, collect device's addresses and names, which are then analysed for statistical purposes.

One of the most common Bluetooth interactions are based on the Bluetooth OBEX profile for sending and receiving files. The interaction is two-away, from the user to the system (e.g. to display a PowerPoint™presentation on a public display) and from the system to the user (e.g. for Bluetooth marketing purposes). The BlueMall system [3] is an example of a Bluetooth-based advertisement system for marketing purposes: users receive marketing information as they enter a specific place.

Other works use the RFCOMM protocol for application specific communications. LectComm, an open-source software, is an application for smart classrooms. Students, equipped with LectComm client applications send their answers to questions and quizzes to the LectComm server application, running on lecturer's computers. Nintendo's Wiimote communication is also based on this type of connections, allowing applications to recognize user's gestures [4] as a way of interaction.

3 Discussion

We propose to build the Bluetooth Hotspot as a middleware component for the development of Bluetooth interaction-based applications. This component should provide the most common Bluetooth services and serve the representative scenarios surveyed in the last section. We now briefly discuss the key design issues of such a component.

The first design issue concerns the integration models with applications which presents two main requirements: (1) how to configure and request the Hotspot services and (2) how to exchange information (Bluetooth sights and files) with applications.

The second key design issue concerns the global state management of the system. This issue is raised because (1) Hotspots are shared between different applications, and thus configuration conflicts may appear, and (2) some scenarios require multiple

Hotspots to cover all the user space in the context of the same application, which may raise global state inconsistency problems.

Finally, considering the existence of additionally Bluetooth services and future application-dependent requirements (for example, Wiimote gesture recognition), the Hotspot architecture should be modular, providing the easy integration of third-party extensions.

4 System Architecture

Our global architecture comprises a central server capable of configuring and managing a set of Hotspots components and the Hotspot itself. The central server is the main point for integrating applications with the Hotspot and for global state managing. The main function of this server is to receive interaction requirements from applications and provide to the Hotspot a set of rules, which model the Hotspot behaviour. This rules describe parameters for scanning intervals, OBEX delivery and OBEX receiver instructions and instructions concerning data exchanging with applications. The Hotspot architecture is depicted in figure 1. It comprises five essential components:

1) the **Bluetooth Modules**, responsible for managing basic Bluetooth functions like device discovering, scanning, and receiving and sending files using OBEX. These modules are initiated by the **Scheduler** module. 2) the **Internal Managers** are responsible for managing the Bluetooth radio interfaces (dongle manager) and the interactions between the Bluetooth component and the applications (rules manager and resources manager). The **rules manager** is responsible for receiving applications configuration rules (possibly from a configuration web site). The **resources manager** is responsible for the communication between the Bluetooth component and the applications. 3) the **Events Channel** implements a decoupled communication model between system components. 4) the **Scheduler** interprets the rules and triggers the correct action for each new event that enters the events channel. 5) a set of third-party **Extensions**, installed by the Hotspot administrator, allowing to extend the system with specific functionality.

5 Conclusions and Future Work

We have presented a short survey on representative Bluetooth-interaction based applications, which are based on the most common Bluetooth services: device and service discovery, Bluetooth OBEX and RFCOMM connection. We support on this study the specification, design and implementation of a Bluetooth Hotspot software component and the corresponding integration models with applications and present our work in progress.

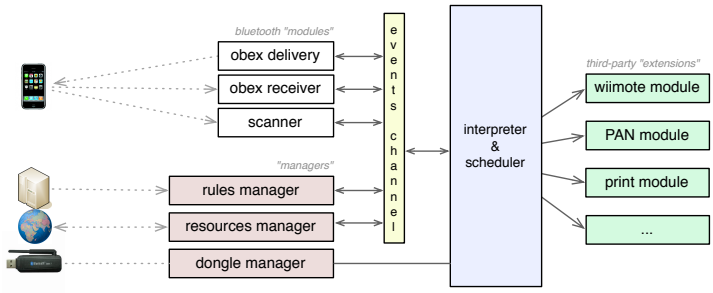


Fig. 1. Proposed component architecture for the Hotspot software.

We have developed a few Hotspot prototypes, which meet partially our architecture. They implement some basic modules such as the scanning, OBEX receiver and OBEX delivery modules. Those Hotspot prototypes are being currently integrated, deployed and shared in the context of different systems.

We are now finishing the implementation of the remaining architecture modules and proceeding to the evaluation of the different deployed systems. We expect to learn from those deployments the necessary lessons in order to validate our main key design issues of global state management, application integration and system modularity. Finally, we expect to evaluate the Hotspot software component in the context of the broader research area of system support for ubiquitous computing.

Given the open nature of the Hotspot component and the natural evolution of Bluetooth interactions, we aim to initiate a process for building the software as an open-source software. We expect to encourage the development of new third-party extensions as well as promote the utilization of Bluetooth as an interaction device in smart spaces.

References

1. Arianna Bassoli Martin Wisniowski Stephan Baumann, Björn Jung. Bluetuna: let your neighbour know what music you like. *ACM*, 2007.
2. Stan Young. Bluetooth traffic monitoring technology. *Center for Advanced Transportation Technology, University of Maryland*, September 2008.
3. José-María Sánchez, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni. Bluemall: A bluetooth-based advertisement system for commercial areas. *ACM*, 2008.
4. Zachary Fitz-Walter, Samuel Jones, and Dian Tjondronegoro. Detecting gesture force peaks for intuitive interaction. *ACM, 5th Australasian Conference on Interactive Entertainment*, 2008.

Indoor Positioning Using a Mobile Phone with an Integrated Accelerometer and Digital Compass

Paulo Pombinho, Ana Paula Afonso, Maria Beatriz Carmo

DI-FCUL, Campo Grande, Edifício C6, 1149-016, Lisboa, Portugal
ppombinho@lasige.di.fc.ul.pt, {bc, apa}@di.fc.ul.pt

Abstract. Although location based applications have been gaining popularity, most positioning devices do not work when in an indoor environment, hindering the development of indoor location based applications. In this paper we propose a technique, based on the detection of footsteps, and the direction in which they were taken by the user, to be able to calculate the position of the user inside a building. We use information about the buildings floor plan to create a graph that can be used to improve the accuracy of the system.

Keywords: Mobile Devices; Indoor Positioning; Step Detection Algorithm.

1 Introduction

Global positioning devices (GPS) are becoming progressively more common in new mobile devices and, for this reason, the real time information about the location of users has become widely used in an extensive range of location based applications. Despite being reliable and precise while in the open, GPS devices need to be able to view a large portion of the sky to correctly calculate the device's position. This renders the GPS useless while indoors. Furthermore, alternative positioning systems like the ones that use GSM or mobile phone tracking do not have enough accuracy to be able to correctly identify a position inside a building.

There are some works that explore indoor positioning mechanisms. Most of these can be divided in three types: the use of infrastructures installed on the buildings, explicitly for indoor positioning; the use of existing Wi-Fi networks; and the use of sensors installed in the mobile device or the user himself.

There are several diverse approaches that use transmitters of some kind, installed on the buildings, and corresponding receivers, carried by the user. Some systems use infrared transmitters [1], RFID tags [2], VHF radio [3], or Bluetooth beacons[4].

Several systems have explored the use of Wi-Fi wireless network access points, and operate by identifying and processing the signal strength information of multiple base stations to triangulate the position of the user (see for instance [5]).

Regarding infrastructure free positioning, Kourogi et al. [6] use sensors placed on the waist of the user, to detect walking stance and velocity. Some approaches use shoe mounted sensors to detect the displacement made by the foot in each footstep and

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 443–446

consequently the displacement made by the user [7]. Finally, Glanzer et al. [8] present a pedestrian navigation system that uses a set of diverse sensors to estimate changes in position and attitude, and obtain the final position of the user.

Despite providing solutions for the problem, the works presented are either based on the existence of an infrastructure in each building, or the need of external sensors placed, for example, on the user's shoes or waist. These sensors are a potential limitation to the natural movement of the user or the practicability of the system. Furthermore, some of the systems require expensive equipments and others, although using cheap beacons, need to install a large number of these to obtain good accuracy.

Since we aim to develop mixed environment (indoor and outdoor) adaptive visualization applications, our goal is to develop an approach that does not need external sensors, beyond those integrated in the mobile device, and that can be used in buildings with no infrastructure installed. Furthermore, since the user is holding the mobile device in his hand, we want an approach that does not hinder the users hand movements, allowing him to retain the freedom of movement they usually have.

In this paper, we present an algorithm that allows the position of users inside a building to be inferred from the movement done by the user. To achieve this goal we use a mobile device with an integrated accelerometer to detect when the users takes a footstep, and a digital compass to determine the direction of the footstep. From this information and the knowledge of the buildings floor plan we can calculate with an acceptable accuracy the indoor location of the user. In the next section we will describe the algorithms and techniques used.

2 Indoor Positioning

We use a 3-axis accelerometer integrated in the mobile phone to capture, in real time, the accelerations that the device is being subjected to.

When in a standing / resting stance, the only acceleration that is present in the mobile device is gravity. If we consider the mobile device to be perpendicular to the floor plane, the X and Z axis would be measuring no acceleration, and the vertical Y-axis would be measuring the gravity, with a value of approximately -9.8 $m.s^{-2}$. Since the users can freely use and move the mobile device, we have no prior knowledge of which accelerometer axis is, at a given point, measuring the vertical acceleration. For this reason, despite potentially adding some noise, we have chosen to analyze the resulting acceleration vector norm, instead of analyzing each axis separately.

When walking, the user will, not only, apply a forward acceleration but also, with a greater magnitude, alternatively apply a vertical upward acceleration followed by a vertical downward one. We have defined four parameters that are used to detect a footstep: a peak amplitude λ_p that represents the minimum positive shift in acceleration caused by a footstep, a trough amplitude λ_t that represents the minimum negative shift in acceleration, a minimum time interval Δt_{min} that needs to go by before completing the footstep pattern and a maximum time interval Δt_{max} that cannot be exceeded for the footstep pattern to be detected. All of these parameters can be changed inside the application in real time.

The step detection algorithm works by constantly checking if the current acceleration is greater than λ_p. When that happens, the application starts to check if the footstep pattern is happening. If a sudden shift in acceleration occurs in less that Δt_{min}, we assume that it was caused by another movement and the step is discarded. If the pattern occurs in more time than Δt_{min} but less than Δt_{max}, and it reaches λ_t creating a pattern similar to the one shown previously, a step is recorded along with the current orientation, which is obtained from the digital compass. If after Δt_{max} we still have not detected the footstep pattern, the step is discarded.

To correctly identify the location of the user inside a building, we use the last known position given by the GPS as the initial position, the information about the steps recorded, the information about the orientation in which they were taken, and the average step length. With this information we can calculate the trajectory of the user and his position. However, to minimize errors caused by steps that are not detected (false negatives), or steps incorrectly detected (false positives), we make corrections to the position of the user through the use of the building floor plan.

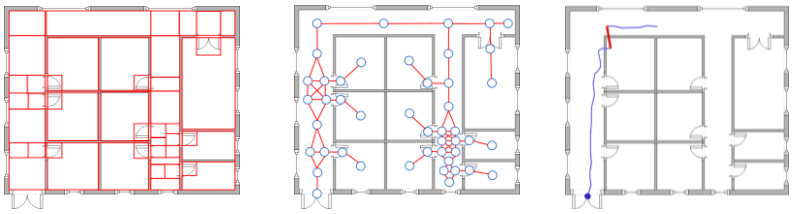


Fig. 1. Definition of the floor plan graph.

We have opted to divide the floor plan in rectangular areas of different sizes, where transition areas (for example, doors) have the smallest size, and areas with no transitions (for example, corridors with no doors) have the bigger areas. Each of these areas corresponds to a node in the graph. Figure 1 shows, on the left, the floor plan with the considered areas in red. In the center figure the graph that was defined is shown, with a node for each area. Finally, in the right figure an example of the path detection and correction is shown. The path the user has taken is drawn in blue over the floor plan and the corrections displayed in a red thick line.

The use of graphs allows us to calculate with each step, the position of the user inside a certain node area. If at any time, the calculated position is outside the current node area, the system will verify if there is a valid transition to another node in the specified direction. If there is, the system calculates the location in the new node area. If there is no valid transition, the system searches for the nearest position where the detected movement would be valid and corrects the user position.

3 Conclusions and Future Work

In this paper, we propose an indoor positioning method that does not need previously installed infrastructures in a building. Furthermore, this approach does not need

external sensors, avoiding the restriction of the user's natural movements when using a mobile device and walking indoors.

In the near future, we need to perform extensive user experiments. These experiments will, not only, allow us to precisely assess how accurate the obtained position of the users is, but also, most importantly, obtain valuable data about the differences in the step patterns originated from a diverse set of users. This knowledge can give us the insight on what the best default step detection parameter values are, and also help us in implementing an automatic calibration of these values. A particularly important parameter, which can cause a high accumulated error, and should thus be automatically adjusted, is the step size, since it varies not only between different users, but also depends on the speed and way the user is moving. Since our aim is to use this algorithm in mixed environment visualization applications, one solution is to use information from the GPS when the user is outdoors.

We also intend to test more complex buildings plans, and the detection of footsteps when going up and down a set of stairs, and the use of escalators and elevators.

Acknowledgments. The work presented here is based on research funded by the FCT - Fundação para a Ciência e Tecnologia through the PTDC/EIA/69765/2006 project and the SFRH/BD/46546/2008 scholarship.

4 References

1. Hiyama, A., Yamashita, J., Kuzuoka, H., Hirota, K., Hirose, M.: Position Tracking Using Infra-Red Signals for Museum Guiding System. In: Proceedings of the Ubiquitous Computing Systems, 2nd International Symposium. pp. 49-61, (2005)
2. Ghiani, G., Paternó, F., Santoro, C., Spano L. D.: A Location-Aware Guide Based on Active RFID's in Multi-Device Environments. In: Computer-Aided Design of User Interfaces VI, Springer London, pp. 59-70. (2009)
3. Ikeda, T., Inoue, Y., Sashima, A., Yamamoto, K., Yamashita, T., Kurumatani, K.: ComPass System: An Low Power Wireless Sensor Network System and its Application to Indoor Positioning. In: Proceedings of the CSTST 2008, pp. 656-662. (2008)
4. Bruns, E., Brombach, B., Zeidler, T., Bimber, O.: Enabling Mobile Phones to Support Large-Scale Museum Guidance. In: IEEE Multimedia, April 2007, pp. 16-25. (2007)
5. Ekahau Wi-Fi Tracking Systems, http://www.ekahau.com
6. Kourogi, M., Ishikawa, T., Kameda, Y., Ishikawa, J., Aoki, K., Kurata, T.: Pedestrian Dead Reckoning and its Applications. In: Proceedings of "Let's Go Out" Workshop in conjunction with ISMAR'09. (2009)
7. Jiménez, A. R., Seco, F., Prieto, J. C., Guevara J.: Indoor Pedestrian Navigation using an INS/EKF framework for Yaw Drift Reduction and a Foot-Mounted IMU. In: Proceedings of the WPNC'10. (2010)
8. Glanzer, G., Bernoulli, T., Wiessflecker, T., Walder, U.: Semi-autonomous Indoor Positioning Using MEMS-based Inertial Measurement Units and Building Information. In: Proceedings of the WPNC'09. (2009)

Engenharia Conduzida por Modelos

UbiLang: Towards a Domain Specific Modeling Language for Specification of Ubiquitous Games

Ricardo Guerreiro, André Rosa, Vasco Sousa, Vasco Amaral, Nuno Correia

Computer Science Department, Faculdade de Ciências e Tecnologia – Universidade Nova de Lisboa, Quinta da Torre, 2829-516 Caparica, Portugal
{rmg15404, adr13041}@fct.unl.pt, vasco.sousa@gmail.com {vasco.amaral, nmc}@di.fct.unl.pt

Abstract. As new ubiquitous projects emerge, it is often required the integration of ubiquitous devices in development frameworks. This commonly leads to developing new frameworks, usually created in General Purpose Languages (GPL). Although this solves immediate problems, it also leads to a decrease of productivity and efficiency, due time spent while adapting code. This results, in most cases, on a development process starting from scratch when most of the times the concepts were already used in previous projects. This project proposes tackling this problem by implementing a Domain-Specific Modeling Language called UbiLang. By carefully taking into account concepts of the domain problem, UbiLang has the main goal of enabling ubiquitous games developers to speed-up their problem specification during the design phase. Allowing then early error detection by validating the system model on a higher abstraction level than code and by improving application development time contribute to faster application prototyping.

Keywords: Domain Specific Modeling Language, Ubiquitous Game Devices, Ubiquitous Gaming, Meta-Modeling, Language Engineering.

1 Introduction

The culture of gaming has a long tradition since ancient board games, such as the Mesopotamian's "Royal Game of Ur"[1]. Currently, games are a real worldwide phenomenon recognized as an important research topics[2] and almost every day new devices are released into market with the purpose of improving gaming experience. Permitting the five human senses to participate in the experience, these improvements have the goal of providing new perspectives of time, space and interaction within the game itself. This technological development can be observed in the evolution of ubiquitous hardware like gloves, head-mounted displays and others, which are often tested in games[2].

As defined in [3], a Ubiquitous Device is – "*an electronic device capable of using its internet, wireless and other networking capabilities that are so embedded in the environment that the devices can be used virtually used anywhere and anytime. This concept embraces a broad range of possibilities, which include communications (cell phones),ubiquitous computing (notebook computers), delivery of images (displays) and products for identifying or managing people and things (objects using wireless IC tags, like RFID tags)*".

Another fact its that the time to market is of a crucial essence in the Ubicomp domain area, due its quick evolution pace. As new Ubiquitous Projects emerge, it is often required the creation of a new types of components (forming new Ubiquitous Devices).

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 449–460

This leads to the necessity of development of new suitable frameworks (that support the project's development) normally programmed in a General Purpose Languages (GPL), such as C++, C# or Java. Although these approaches tend to solve immediate problems, they also lead to a decrease of productivity, due to the low-level code re-use of solutions that are very similar. This compels, in most cases, to development from scratch, as already analyzed in [4].

At the same time, the domain expert hardware/game developer would benefit from a language with a higher level of abstraction that would allow the specification of models using domain terminology instead of a language from the solution domain (e.g., C++ or Java). The last situation of using code, potentially leads to semantic gap problems (i.e., the way we think about the problem in terms of games, has to be twisted, in an error prone fashion, to the way it is expressed in computational terms).

A Domain-Specific Modeling Language (DSML) is according to [5]: a language "*used to make specifications that manual programmers would treat as source code and if formed correctly it should apply terms and concepts of a particular problem domain*". We propose a DSML to solve the referred problems, by allowing the development of applications with Top-down (Designers to Programmers) or Bottom-up (Programmers to Designers) paradigm views.

To accomplish this we will, in section 2, engage in a domain analysis of the Ubicomp topic by exploring reusable aspects of the hardware components and correspondent behaviors. After that, a design of the DSML will be made, which is expected to provide the domain expert with faster means for lowering error prone model specification in section 2 and 3. This will be achieved by dividing the gained knowledge into two levels:

I. **Structural Level**: where all the hardware components and the most used virtual objects properties/aspects are gathered;
II. **Behavior Level**: where the most used application behavior transitions and states are gathered;

Finally, in section 4, a solution will be provided using a real life case-study of a wearable interface developed by the Multimedia Group at FCT-UNL dubbed as Gauntlet [6] and an application example for the Gauntlet, named Noon [6, 7]. We will then validate our statements of productivity increase and usability through an empirical study, using domain experts as subjects. In section 5 we conclude our work.

2 Domain Background

Ubiquitous Projects belong to the Ubiquitous Computing (Ubicomp, for short) domain area and, depending on the context, to the larger Multimedia base domain area. This allows in addition the Ubicomp are to be fused with another domain area, the Augmented Reality (AR), due to the latter using the same technology developed in Ubicomp to expand reality with computer interaction. These domain areas, similar as they are, can have the same development background. So the next sub-sections will introduce the most relevant development methods for these domain areas and conclude by presenting the domain analysis made with the Ubiquitous Projects.

2.1 Augmented Reality

As stated in [8]: "*AR is a variation of Virtual Environment (VE), or more commonly known as Virtual Reality (VR)*". A VR differs from an AR, as the first merges the user perspective with the artificial world, therefore hiding the real world. The AR approach

uses computer generated elements to complement reality in real-time. To achieve results like this, an AR project can use Ubicomp technology (such as a PC with a camera and a specific pattern, for example) and build applications able to generate these VEs. These applications are then called AR applications and are commonly developed using AR frameworks, which have the ability to manage the particular Ubiquitous Devices used in the application. Other developments methods can consist on Visual Programming Languages (VPL) and GPLs. These three concepts are going to be further discussed.

Augmented Reality Frameworks, can go from software libraries like ARToolkit [9] to concepts of collaborating distributed services like the Distributed Wearable Augmented Reality Framework (Dwarf for short) case [10, 11]. Still, frameworks like ARToolkit, due to the fact of being "software libraries"[12], when compared with Domain-Specific Modeling (DSM) or VPLs, can miss the major advantages of the fast application prototyping inherit in these latter approaches. It is, for example, also surpassed by the Dwarf framework *services*, however as a gain, its modules have already been validated and tested and therefore can be reused/applied in a possible future Domain Specific Modeling (DSM) Generator approach. In Dwarf's architecture, because of its concepts of collaborating distributed "*services*" that can be developed in a wide range of programming languages (e.g. Java, C++, Python), give the framework the power of platform/programming language independency. These services consist in collections of interdependent modules of code that have a set of requirements called "*Needs*" and capacities called "*Abilities*". Each of the modules is then connected with other modules within a network, forming groups that are then controlled by a special "*service*" named Service Manager. This "cluster" concept is similar to an element abstraction in a DSML, yet it lacks the domain specification possible by such elements as it does not reuse the code as a DSM Generator would do, thus keeping the same issues when compared with GPL approaches.

Visual Programming Languages (VPLs), are programming languages that let programmers create new applications by graphically manipulating program modules. A VPL, to achieve that purpose, uses visual expressions, spatial arrangements of text and/or graphic symbols, rather than specifying them in a textual manner. So a VPL oriented environment results in a language with an inherent visual expression for which there is no obvious textual equivalent [13]. They are also associated with specific applications or frameworks, with some examples like: Max/MSP/Jitter [14], Pure Data [15], and others [4].

Media Processing Frameworks is a type of development where the programming is still being made in a textual manner using GPLs for that purpose. ARToolkit, for example, can also be classified as a Media Processing framework or software library that uses GPL. Some widely used examples are Processing [16] and openFrameworks [17]. Both present a simplified interface to powerful libraries for media, hardware and communication interaction.

These solutions, despite solving the prompted issues (multimedia object manipulation, for example), do not address all Ubicomp problems (as the majority of the Ubiquitous Projects do not take a framework approach to solve their problem, but instead use GPL programming). This divergence comes as the Ubicomp domain area and the AR domain

area can be fused, there are some components/requirements or actions in Ubiquitous Projects that are not commonly taken into the AR domain scope. Also, in Ubiquitous Projects, there exists a tendency to develop new types of devices, from different off-the-shelf components with the main objective of innovating how players interact within a particular game. AR Projects instead, normally take already developed devices and just enhance the user's perception of reality. In the VPLs case, being similar to a Graphic DSMLs as they are, the biggest discrepancy comes from the capability of a DSML to be built specifically to a particular domain. Finally, GPLs and Media Processing Frameworks case still have reusability issues from one application to another together with possible code errors or/and inconsistencies between both programmers and application designers visions for the final application. Still this analysis demonstrated the most common components and functionalities currently available, providing an idea of the ones that were most used or required by the AR community in general.

2.2 Survey of Projects

Presently there is a wide plethora of Ubiquitous Projects and some of them were analyzed in surveys like [2, 6]. With the objective of widening the domain scope to demonstrate the usability of the proposed DSML in a wide range of Ubiquitous Projects, a selection parameter was required to analyze these projects. The analysis began by searching for similarities between projects. This lead to two major comparison parameters: (1) the hardware components used to build the particular Ubiquitous Project and (2) the way these components were used or the behavior they add in the program workflow. With the first comparison parameter it was possible to restrict the analyzed projects to a much smaller number. The second parameter came when even, repeating some hardware components, the behaviors each of them had (in their respective projects) was different from one application to another. This gave not only different and new behavioral analysis (which was useful to be generalized) but decreased even further the number of projects to be analyzed. With these criteria in mind, we have proceeded to a discussion and analysis found in [18] of the various selected projects that were found to incorporate the widest array of components and behaviours: 6th Sense [19], Blinkenlights [20], Brainball [21], InStory [22], Pirates! [23], Uncle Roy All Around You [24], Epidemic Menace [25], Noon [6] and Headbanger Hero[26] .

The analysis then gave an wide range of the commonly used components in the domain area. They were then added to the lowest levels of the Structural Level and soon it was clear that, in order to offer some type of organization at modeling time, the components were needed to be divided in several categories, as seen in Fig. 1. This allowed programmers/hardware developers to rapidly and easily construct devices and therefore a concept of a "Virtual Device" was added to the language.

Within each of the categories, shown in Fig. 1, it was put the identified elements of the analysis. For example, in the "Visual" category it was needed a "Monitor", "Camera" and "Visual Effects" elements, which enabled the definition of a generalized input, output hardware component and also a property that gave effects to all visual components, (being also possible to define individually "Visual Effects" element within each defined "Monitor" and "Camera" element), the other categories were also filled with identified domain elements abstractions and more details can be found in [18].

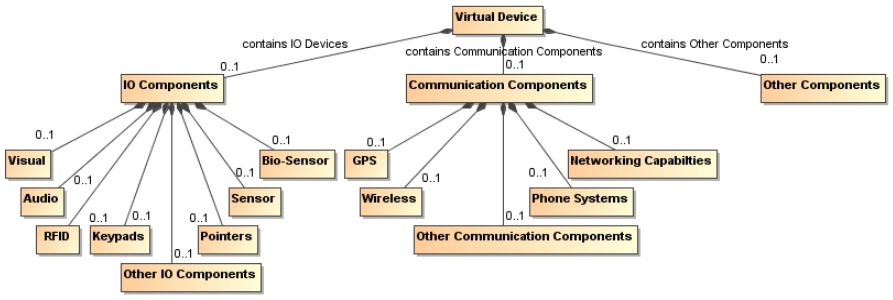


Figure 1. Virtual Device Compartments

Additionally to the components used in the projects, their contexts were also analyzed, resulting in some relevant concepts. One of these, was that the need (in the majority of the cases) to interact with virtual objects which permitted, for example, interaction with physical hardware components/objects external to the application itself (audio, video, textual, video file, or even a model generated by an outside application). With this notion, the Structural Level was completed by creating a new element called "Object" which stood at the same hierarchical level of the Virtual Device. To this "Object" element was given the possibility to define virtual representations, which could be referenced to a specific type of external multimedia or model representation files. Furthermore, it could have its own physical properties, in case of a NPC or human avatar, or even virtual values attributes like health, for example. The relation with the Virtual Device would then be made via its physical representations properties, as for example RFID Tags, real world coordinates or association with specific Virtual Devices.

From the projects, another identified requirement/concept was the need for (at modeling/application designing time), provide users with the opportunity to instantly work on the application's workflow without concerns and therefore rapidly express (in a language much closer to their own) a prototype of the application's behavior/context. So with this concept and the analysis already done, the Behavior Level began to be developed following a template of a finite state machine model, for all the identified actions [18].

2.3 Model Driven Development

In a Model Driven Development (MDD) approach, specifically in DSM, a model, for short, serves as a mean of visually representing abstractions of system concepts and features [5]. This model can be used for designing purposes (when the problem domain requirements are too complex to be expressed with code), to reverse engineer systems (by creating model-based visual documentation, which helps understand the capabilities of a system after it was already designed and built) and other functionalities. A metamodel is a DSML specification, which describes an abstraction level higher than the model, at the point that it abstracts the concepts used in the model or abstract syntax below. The metamodel has the purpose of describing and expressing concepts of a language, their properties, constructions and rules (like relationship, correctness or hierarchy rules between concepts) [4, 5, 27].

Domain Specific Modeling, has two aims: (1) raise the level of abstraction further than the current programming languages by specifying the solution using problem domain concepts; (2) generate final applications in a chosen programming language or other form of high-level specification [5].

The first objective specifies a Domain Specific Modeling Language, or DSML, which is a kind of typically declarative language that gives an expressive power to a particular problem domain, simplifying the development of generalized applications in these specialized domains using high-level abstractions of the domain concepts for that purpose. The definition process of a DSML, as stated in [5], consists of five main phases: (1) Domain Analysis where the problem domain necessities and/or requirements are identified and analyzed based on a number of different applications/systems that belong to the same domain for similar features or concepts; (2) Language Design where the domain analysis features gathered from the previous phase are formalized in the design of the DSML metamodel, (3) Language Implementation represents the definition of the visual representations of the DSML features; (4) Language Testing validates a DSML to multiple example cases and (5) Language Maintenance when the domain area evolves, leading to new requirements that will be used to update the language.

In the second objective the generation process can, normally, be supported by an application framework or API using a domain specific generator or a high level of abstraction. The idea behind a DSM generator comes from the notion that in DSM, application modelers do not expect the full code of the framework to be implemented but only the "code" they modeled in the DSM Editor instead [5, 28, 29].

DSM solutions are meant to be used when applications features or domain requirements have similarities. In these situations, application programmers tend to focus in their applications unique features development rather than reimplementing similar functionalities [5]. In addition, when comparing with other general purpose modeling languages, like UML [30] (Unified Modeling Language), the DSM process takes a large advantage as it can join all the diagrams needed in the UML development process, for instance, into a single metamodel. By the time the models are being implemented, in the UML process, they are made in a independent way from the designed models themselves. This can lead developers astray with questions of not properly specified aspects that are (possibly) not true in the application/system domain or were just simply ignored. Also it is virtually impossible to generate full application's code as the generality of the modeling language does not know anything of the application domain origins or its problems. In this way, in the DSM process, the metamodel, prevents semantic errors in illegal designs that do not follow the model architectural rules defined previously. Subsequently, the code generated from it does not contain logic errors, syntax or careless mistakes as it was specified by a DSM Generator (that was developed by an experience domain expert developer) [4, 5, 29, 31].

3 UbiLang

Taking the five domain phases described in the previous section, this chapter presents the work done in the development of UbiLang.

Domain Analysis, where the elements, classes and concepts were identified as requirements to be available in the DSML, this was done in Chapter 2 with the analysis in the various domain areas and projects. One these concepts, was the need of providing at modeling time, the two programming paradigm views (Bottom-Up (programmers to application designers) and Top-Bottom (vice-versa)). This came of the necessity to not

constrain the creativity in the development process and allowing designers to work on the application behavior first and component configuration second (Top-Bottom view). Also the programmers could configure (if desired) each of their application components (hardware or virtual) and leave the application behavior formalization for afterwards (Bottom-Up). With this notion in mind, it was defined the two levels already explained, within the DSML: Structural Level and Behavior Level.

Design, the domain experts, independently from the development approach, categorized their applications between several categories (visual, audio, keypad, etc.) and behavior flows. With analysis done in Chapter 2 (previous phase) came too many elements/concepts, so a filter was added that abstracted several elements/actions into one. An example, already described, was the elements in the visual category, the other categories went through the same process and more details can be found in [18]. This type of definition was adopted in UbiLang, because it allowed an easier device's expression and greater creativity for programmers and designers alike, taking it to full extent levels without concerns about programming new hardware components/behavior flow with error-prone development.

Also the analysis acknowledged the most relevant properties required for each of the identified elements/connections (for configuration purposes). For example, in the specific comparison connections cases, it was identified the most common used comparison parameters. These parameters went from different source-target types to comparison conditions, as for example positional parameters between components and objects GPS position. Furthermore it was added the possibility, (for all connections types), to be delayed a certain amount of time desired by the modeler and displaying this information in each of the textual label's connections alongside its name[18].

The next Fig. 2 presents a general "metamodel" created from the previous explained phases, with the concerns for the modeler creativity, for each game it was added the possibility to organize it using several "Virtual Device Manager", "Object Manager" and "Behavior" elements. This gave the possibility to, for example, organize devices through teams (human and NPCs), virtual objects through specifications (avatars and virtual objects) and behaviors through specific actions (repeat actions and one time actions). The "LAN" element stood at the same level as the Virtual Device as it represented the network itself as well the Virtual Devices (they still required the "LAN Capability" element [18]) connected to it.

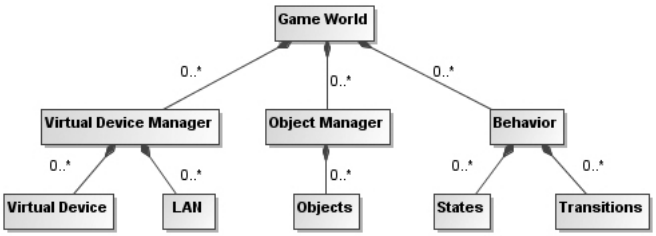


Figure 2. General UbiLang metamodel.

Implementation, the language was implemented using the workbench Eclipse with the aid of the Graphical Modeling Framework/Ecore Modeling Framework (GMF/EMF)[32] and its plug-ins. This workbench offered one of the best possibilities to customize elements interface layouts and other graphical interface improvements, when compared with another tools [29].

As seen in Fig. 4 to 5, the workbench offered the possibility of designing basic elements by displaying visual information in form of icons in addition to the textual

labels. As well it offered a way to encapsulate information, in the form of compartments, this is helpful when the models tend to get bigger and more complex, as in the case of Fig.5. Also, for aiding the user, when building their model there was various assistants, like a customizable side palette toolbar that had all the elements, which the user could create its models. When hovering the canvas, or the inside of a compartment, a popup bar displayed the possible elements that could be inserted. In the Behavior model by clicking and hovering on the border of the icons/compartments, two arrows displayed (an arrow going out and another going in), and clicking in either one would displayed all the possible connections to/from the clicked element to an existent/new element. Also, in the Behavior model compartments, the state elements were grouped in 4 categories: General Actions (general application behavior as new threads, initialize a behavior or another specific behavior), General Object Actions (change configurations of an object, positions, etc), General LAN Actions (general networking actions as send files to all the groups in the network) and General Virtual Device Actions (change devices configurations). These compartments, for an easier use, when double-clicked on the border opened a new canvas that allowed an easier model design and when closed displayed the edited elements of the second canvas, in the inner compartment, with the opposite still possible. Finally, when right-clicking on each of the elements placed on the canvas (icons, connections or compartments) and displaying (if not already) the built-in Eclipse's "Properties View" plug-in, it was possible to change the properties identified of the domain analysis and associated with each of the elements[18].

This visual proposal [18] was found to be the most conformable to the objective of combining both programming view paradigms and also offer an easy and understandable way of designing ubiquitous game applications, thus allowing the creative process to not be hindered with programming issues.

4 Validation

Continuing the DSM process the next phase, validation, was done by modeling an application called Noon. Developed by Tiago Martins, the Noon's context introduced a long-story mystery were the player took the role of a detective [6, 7], using a device called Gauntlet, seen in Fig. 3, and another called Tome. This was one of the case-studies chosen to validate the language.

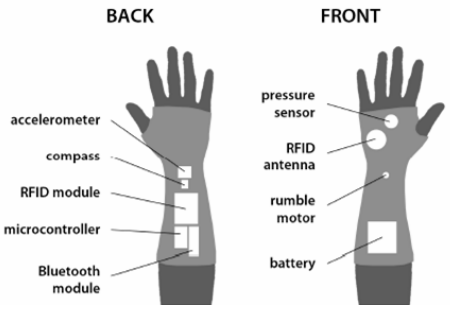


Figure 3. Gauntlet, extracted from [6].

Components, The choice of hardware technologies relied in a selection procedure based primarily on wearability factors (how the hardware influences motion responses, muscular shifting, the temperature it causes and the total weight it brings to the user) and secondly by how the sensors reacted to different types of environment. The final ubiquitous devices technology composition for the Noon framework was for the:

- Gauntlet - with a possible representation seen in UbiLang on Fig. 4 with an accelerometer, a digital compass (or magnetometer, which combined with the accelerometer can provide a more absolute measure of the user's tri-axial arm movement), a RFID Antenna and Module (combined can read and interpret RFID Tags), a Bluetooth module (used for communications with the Tome, a force resistance (or pressure) sensor, a rumble motor and a LED, for user feedback and a battery (for portability).

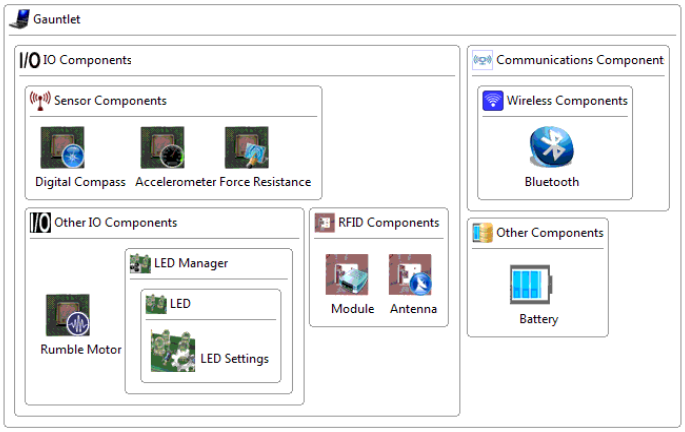


Figure 4 – A possible interpretation of the Gauntlet in UbiLang.

- Tome – this object can be any platform of Ubiquitous Computing technology (a notebook, a PDA, a smart-phone, etc.). The only mandatory requirement is a monitor, a set of speakers and a Bluetooth module for receiving Gauntlet's communications.
- Object Tags – Each of the in-game objects has a RFID Tag associated with it which the Gauntlet reacts upon.

Virtual Objects, within the game there were a total of six objects (associated with the physical tags mentioned above) that react with different time periods within the game depending on the direction where the Gauntlet is pointing: a snow globe, a cup, a picture, a schoolbook, a hammer and a table clock.

Behavior, the Noon application behavior starts for waiting any of the user input. If the user puts the Gauntlet in a vertical position, the accelerometer and digital compass detect this movement and display a visual feedback by turning on the Gauntlet's LED. The user can also make a horizontal movement, to which the application reacts by giving a clock ticking sound and permitting to change the "game time". The user, then uses the Gauntlet's ability of reading RFID tags (which are attached to physical objects), by means of the RFID Antenna. This allows "triggering memories" from a game character called Mrs. Novak, and display them on the Tome. In some of these objects, if it is detected a specific motion pattern, for example "shaking the snow globe" or "pouring the cup", the player is shown "deeper memories". In the process, a more "intense memory" triggers a Poltergeist, which the player must capture by listening to the sounds it makes, in order to continue the investigation and solve the mystery[7]. Noon is also an endless game so when all the memories have been read, the game restarts. Fig. 5 represents a proposal for a portion of the applications behavior (capture of the poltergeist) in UbiLang.

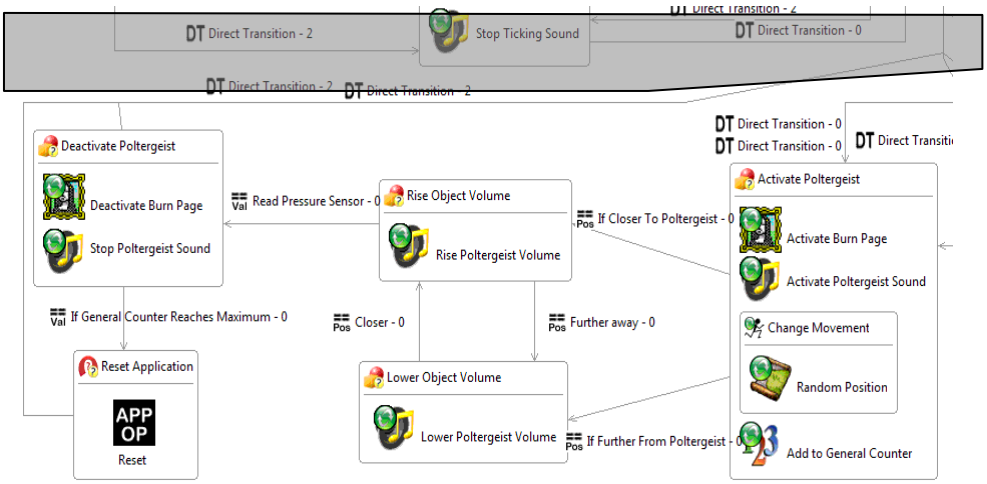


Figure 5. A partial view of an interpretation of the Noon Behavior

After sucessfully modeling a use-case the second part of the validation process was letting the domain experts test the language themselves. By collecting a group of 9 users, as advised in [33-35] and giving instructions to install the plugin, it was asked to them to complete a different exercise that had a different context from Noon, which they started in a provided tutorial and continued to answer a questionnaire. Some of the responses were:

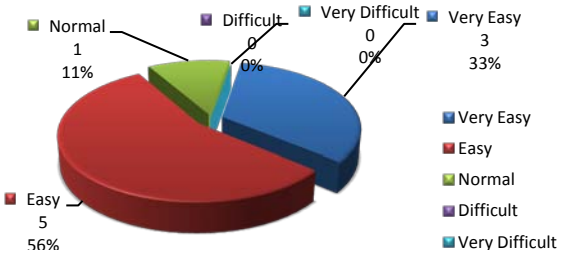


Figure 6 – "How easy it was to learn the UbiLang concepts?"

Other question "Do you feel UbiLang is a value-added compared to the previous application designing system?" the users fully agreed that it was a value-added to a coding development system alternative or adaptation of VPL's or Media Processing Frameworks. The follow-up question "Explain why?", provided such answers like: "*It might allow people with no imperative programming experience to experiment with behaviors and produce programs for Ubicomp interfaces*" and "*Using a visual representation allows a better understanding of the application and it is easier (less errors) and faster to build it. Furthermore by using models we gain various advantages, instant verification, possibility to simulate, automatically generate code and so on*".

5 Conclusions and Future Work

By means of case-studies and user assertions, we demonstrated that we reached our objectives and, in addition, the ubiquitous models maintenance time is also greatly

shortened by the simple act of adding/removing a missing/existent component or action. The language is but a first step on the development as it is ready to be taken to a DSM Generator phase by assigning modules of validated code to the language elements and quickly pass from a designed model to executable code and therefore running application. Once we have a complete DSL, we can also explore model verification techniques to detect inconsistencies in the implementation already at design time. In what concerns the language editor, it would be interesting to enhance it so it would support new visualization modes and provide a better usability for the users. The main objective of developing this language was to provide an easier and faster way to increase efficiency in the creation of Ubiquitous games. By quickly making each of the diagrams types, it is easier for the end-user to validate if a desired application is feasible. This allows managing what are the required components and configurations at an appropriate level of abstraction in a domain expert using terminology of the domain instead of just computational terms.

References

1 Harold James Ruthven Murray: 'A History of Board-Games Other Than Chess' (Gardners Books, 1969. 1969)

2 Tiago Martins, Nuno Correia, Christa Sommerer, Laurent Mignonneau: 'Ubiquitous Gaming Interaction: Engaging Play Anywhere', in Heidelberg, S.B. (Ed.): 'The Art and Science of Interface and Interaction Design' (Vol.14, pag. - 115-130; Springer Berlin / Heidelberg, 2008)

3 http://www.hitachi.com/rd/sdl/glossary/u/ubiquitous_device.html, accessed 10th of July 2010

4 André Rosa: 'Designing a DSL solution for the domain of Augmented Reality Software'. Masters in Computer Sciences, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa, 2008/2009

5 Steven Kelly, Juha-Pekka Tolvanen: 'Domain-Specific Modeling: Enabling Full Code Generation' (John Wiley & Sons, Inc, 2008)

6 Tiago Martins, Teresa Romão, Christa Sommerer, Laurent Mignonneau, Nuno Correia: 'Towards an Interface for Untethered Ubiquitous Gaming'. Proc. 2008 International Conference on Advances in Computer Entertainement Technology, Yokohama, Japan

7 http://tiagomartins.wordpress.com/projects/noon-a-secret-told-by-objects/, accessed 10th of July 2010

8 Ronald T. Azuma: 'A Survey of Augmented Reality', Presence: Teleoperators and Virtual Environments, 4 August 1997, 6, pp. 355-385

9 https://launchpad.net/artoolkit/, accessed 10th of July 2010

10 http://ar.in.tum.de/Chair/ProjectDwarf, accessed 10th of July 2010

11 Prof. Bernd Bruegge Ph.D., Prof. Gudrun Klinker, Ph.D.: 'DWARF - Distributed Wearable Augmented Reality Framework'. Proc. Chair for Applied Software Engineering, Technische Universitat Munchen

12 Martin Bauer, Bernd Bruegge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Stefan Riß, Christian Sandor, Martin Wagner: 'Design of a Component–Based Augmented Reality Framework'. Proc. Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on, New York, USA 2001

13 Wesley M. Johnston, J. R. Paul Hanna and Richard J. Millar: 'Advances in dataflow programming languages'. Proc. ACM Computing Surveys (CSUR), New York, NY, USA March 2004

14 http://www.cycling74.com/products/max5, accessed 10th of July 2010
15 http://puredata.info/, accessed 10th of July 2010
16 http://processing.org/, accessed 10th of July 2010
17 http://www.openframeworks.cc/, accessed 10th of July 2010
18 Ricardo Guerreiro: 'A DSML for Specification of Ubiquitous Games'. Masters in Computer Science, Faculdade de Ciênicas e Tecnologia, Universidade Nova de Lisboa, 2009
19 http://www.pranavmistry.com/projects/sixthsense/index.htm, accessed 10th of July 2010
20 http://www.blinkenlights.net/, accessed 10th of July 2010
21 Sara Ilstedt Hjelm: 'Research + Design: the making of Brainball'. Proc. Interactions 2003 pp. Pages
22 Nuno Correia, Hélder Correia, Luís Alves, Luís Romero, Carmen Morgado, Luís Soares, José C. Cunha, Teresa Romão, A. Eduardo Dias, Joaquim A. Jorge: 'InStory: A System for Mobile Access, Storytelling and Gaming Activities in Physical Spaces'. Proc. ACM SIGCHI - International Conference on Advances in Computer Entertainement Technology, Universidade Politécnica de Valência, Valência, Spain 2005
23 Staffan Bjork, Jennica Falk, Rebecca Hanson, Peter Ljungstrand: 'Pirates! Using the Physical World as a Game Board'. Proc. Interact 2001, Tokyo, Japan 2001
24 Steve Benford, Martin Flintman, Adam Drozd, Rob Anastasi, Duncan Rowland, Nick Tandavanitj, Matt Adams, Ju Row-Far, Amanda Oldroyd, Jon Sutton: 'Uncle Roy All Around You: Implicating the City in a Location-Based Performance'. Proc. International Conference on Advances in Computer Entertainement Technology (ACE) 2004, Singapore
25 Irma Lindt, Jan Ohlenburg, Uta Pankoke-Babatz, Wolfgang Prinz, Sabiha Ghellal: 'Combining Multiple Gaming Interfaces in Epidemic Menace'. Proc. Conferences on Human Factors in Computing Systems 2006, Montréal, Québec, Canada
26 http://www.headbanghero.com/, accessed 10th of July 2010
27 Vasco Sousa: 'Model Driven Development Implementation of a Control Systems User Interfaces Specification Tool'. Masters in Computer Sciences, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa, 2008/09
28 Arie Van Deursen, Paul Klint, Joost Viser: 'Domain-Specific Language: A Annotated Bibliography': 'ACM SIGPLAN NOTICES' (Vol. 35, Issue 6, pag. 26-36; ACM, New York, USA, 2000)
29 Vasco Sousa, Vasco Amaral and Patrícia Conde: 'Towards a full implementation of a robust solution of a Domain Specific Visual Query Language for HEP Physics analysis'. Proc. Computing in High Energy and Nuclear Physics (CHEP) 2007
30 http://www.uml.org/, accessed 10th of July 2010
31 Krzysztof Czarnecki: 'Overview of Generative Software Development'. Proc. Unconventional Programming Paradigms (UPP) 2004, Mont Saint Michel, France
32 EMF: http://www.eclipse.org/modeling/emf/; GMF: http://www.eclipse.org/modeling/gmf/, accessed 10th of July 2010
33 Jakob Nielsen and Thomas K. Landauer: 'A mathematical model of the finding of usability problems'. Proc. ACM INTERCHI'93 Conference, Amsterdam, Netherlands, April 1993
34 Pedro Gabriel: 'Software Languages Engineering: Experimental Evaluation', Faculdade de Ciênicas e Tecnologia, Universidade Nova de Lisboa, 2009
35 http://www.useit.com/alertbox/20000319.html, accessed 10th of July 2010

Web-Application Modeling With the CMS-ML Language*

João de Sousa Saraiva, Alberto Rodrigues da Silva

INESC-ID / Instituto Superior Técnico
Rua Alves Redol, 9, 1000-029 Lisboa, Portugal,
joao.saraiva@inesc-id.pt, alberto.silva@acm.org

Abstract. The Model-Driven Engineering paradigm has become increasingly popular due to its advocation of using models as first-class citizens in the software development process, while artifacts such as documentation and source-code can be produced from those models by using automated transformations. On the other hand, we are currently witnessing the rise in popularity of a particular kind of web-application, Content Management Systems (CMS). This paper overviews the CMS Modeling Language (CMS-ML), a graphical language for the high-level modeling of CMS-based web-applications. CMS-ML is oriented towards enabling non-technical stakeholders to rapidly model a web-site supported by a CMS system. The language also allows for its extension, in order to support the modeling of more complex web-applications.

Resumo O paradigma da Engenharia Conduzida por Modelos tem-se popularizado devido à sua utilização de modelos como cidadãos de primeira classe no processo de desenvolvimento de software, enquanto artefactos como documentação e código-fonte podem ser produzidos a partir desses modelos através de transformações automatizadas. Por outro lado, estamos actualmente a assistir à ascensão de um determinado tipo de aplicação-web, os Sistemas de Gestão de Conteúdos (CMS). Este artigo apresenta o CMS Modeling Language (CMS-ML), uma linguagem gráfica para a modelação a alto nível de aplicações-web baseadas em CMS. Esta linguagem tem como objectivo permitir que os interessados não-técnicos possam rapidamente modelar um web-site suportado por um sistema CMS. A linguagem também permite a sua extensão, de modo a suportar a modelação de aplicações-web de maior complexidade.

* This work was supported by FCT (PhD Scholarship SFRH/BD/28604/2006 and INESC-ID multiannual funding) through the PIDDAC Program funds.

1 Introduction

The global expansion of the Internet has led to the appearance of multiple web-oriented Content Management Systems (CMS) [1,2] platforms. CMS systems are web-applications oriented towards the dynamic management of web-sites and their contents, providing concepts such as User, Role, Language, WebComponent, Dynamic WebPage and Visual Theme [3,4]. These systems typically present aspects such as extensibility and modularity, independence between content and presentation, support for several types of contents, support for access management and user control, dynamic management of layout and visual appearance, or support for workflow definition and execution.

Development of web-applications supported by CMS platforms is usually done via traditional development processes, in which source-code is the primary artifact, and design models and documentation are considered only as support artifacts. As is already well-known in the Software Engineering community, such processes are typically time-consuming and error-prone, because they rely heavily on programmers and their execution of repetitive tasks. Also, the source-code and the design models are often out of sync, because changes to source-code are not automatically propagated to the models.

On the other hand, Model-Driven Engineering (MDE) [5] development processes consider models as the primary artifact, and other artifacts (such as source-code or documentation) are produced automatically from those models via automatic model transformations. Besides leaving most of the repetitive tasks to those transformations, these processes present additional advantages, such as: (1) relieving developers from issues like underlying platform complexity or inability of programming languages to express domain concepts; or (2) targeting multiple deployment platforms without requiring several different code-bases.

In this paper we present the CMS Modeling Language (CMS-ML), a graphical modeling language oriented towards the high-level modeling of CMS-based web-sites and web-applications. CMS-ML has a number of aspects that distinguish it from other web-engineering-oriented modeling languages and approaches, namely: (1) it is CMS-independent, and so it does not address implementation details; (2) it allows language users to extend it (albeit in a controlled manner) with new concepts; and (3) it is meant to allow regular stakeholders (e.g., users not aware of software development problems) to easily understand and change the model. This language was developed within the context of our research regarding the usage of multiple modeling languages to address the various stakeholder perspectives of a web-application's development [6].

The remainder of this paper is structured as follows. Section 2 provides a brief overview of our approach for the development of CMS-based web-applications, in the context of which CMS-ML was created. Section 3 presents the CMS-ML modeling language, as well as the underlying metamodeling rationale. Section 4 presents a discussion of CMS-ML and our approach, and compares it with some related work. Finally, Section 5 presents the conclusions for our research so far as well and points out some future work.

2 Context

The CMS-ML modeling language was created in the context of our proposed model-driven approach for the development of CMS-based web-applications [6], which is illustrated in Figure 1. Instead of defining a single CMS-oriented modeling language, our approach defines two languages: (1) CMS-IL (CMS Intermediate Language), a common low-level language for CMS platforms; and (2) CMS-ML, which provides a set of elements that are used to quickly model a typical web-application.

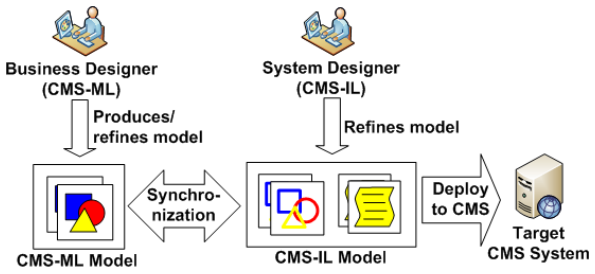


Fig. 1: The proposed MDE-oriented approach.

The Business Designer (a generic term to identify non-technical stakeholders) creates a CMS-ML Model that represents the intended web-application, according to some predetermined business requirements. After applying an automatic transformation from that CMS-ML Model (and obtaining a corresponding CMS-IL Model), the System Designer determines whether that CMS-IL Model is satisfactory, namely by identifying any particular requirements that could not be addressed by CMS-ML alone; if any such requirements exist, the obtained CMS-IL Model must be modified/refined by the System Designer to address them. After this refinement, the CMS-IL Model should be an accurate (and correct) representation of what the intended web-application should be. This CMS-IL Model is then deployed onto a target CMS in one of two ways, depending on the CMS: (1) importation to a CMS Model Interpreter component, or (2) generation of low-level artifacts and subsequent installation. The first alternative is preferable, as it only requires that a CMS Administrator (with administrative privileges) upload the CMS-IL Model into a CMS Model Interpreter, but it will not be feasible in CMS platforms that do not have that component available. In such cases, the second alternative (not illustrated, for simplicity reasons) requires the intervention of a software developer – to perform the compilation of the generated artifacts – and of a CMS Administrator, in order to both deploy the compiled artifacts and make any necessary configuration changes.

This paper will not describe the approach further, as it has been described in [6], and the main objective of this paper is to present the CMS-ML language in greater detail.

3 The CMS-ML Modeling Language

The CMS Modeling Language (CMS-ML) is a graphical modeling language for the high-level specification of CMS-based web-sites and web-applications. Its main objective is to allow regular non-technical stakeholders to look at a web-site's model, *understand* it, and *make changes* to it.

CMS-ML modeling is focused on two different (and complementary) types of model, (1) Web-Site Templates and (2) Toolkits. A *Web-Site Template* (or just *Template*) is a model that reflects the intended web-site's structure and behavior; this Template is modeled using *CMS elements* – such as Role, DynamicWebPage, WebComponent – that are provided by CMS-ML. On the other hand, a *Toolkit* allows the addition of new modeling elements to the set of CMS elements that are available for modeling a Web-Site Template, in a controlled and reusable manner (due to text size constraints, we will not be going into detail regarding the metamodeling rationale behind this language extension capability).

3.1 Roles

Because of the multiple web-site and web-application concerns that CMS-ML addresses, the modeling effort for creating Web-Site Templates will be divided among different kinds of roles, according to the "separation of responsibilities" principle. CMS-ML considers the following modeling roles, depicted in Figure 2: (1) the *Toolkit Architect*, who specifies Toolkits; (2) the *Web-Site Template Creator* (usually just called "Template Creator"), who models a Web-Site Template; (3) the *Web-Designer*, who defines visual themes and graphics for the Template; and (4) the *CMS Administrator*, who instantiates the elements defined in the Template. Of these roles, the most relevant are the Toolkit Architect and the Web-Site Template Creator. The remainder of this section will present an overview of their modeling tasks.

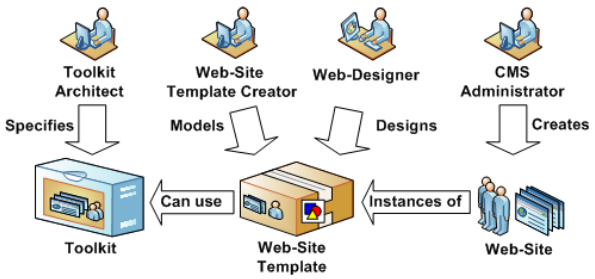


Fig. 2: The modeling roles and artifacts considered by CMS-ML.

3.2 Web-Site Template Modeling

CMS-ML provides a set of generic modeling elements (generically called "CMS elements") that Web-Site Template Creators can use to define their Templates for CMS-based web-sites. A Template is defined according to a set of views (illustrated in Figure 3): (1) the *Structure view*, which specifies the web-site's structural components; (2) the *Navigation view*, specifying the possible navigation flows between the structural components of the web-site; (3) the *Roles view*, which deals with the set of responsibilities that the web-site expects its users to assume; (4) the *Permissions view*, specifying which Roles have access to the web-site's structural components; (5) the *Users view*, which specifies particular CMS users that are considered fundamental to the modeled web-site; (6) the *Languages view*, which deals with internationalization and the languages that the web-site should have available; (7) the *Contents view*, which specifies contents (e.g., pieces of text) that should be available on the web-site; and (8) the *Visual Themes view*, which specifies graphical parameters about how users should view the web-site. The "bootstrapping views" are separated from the other views because they are not necessary for the modeling of a web-site. Instead, the bootstrapping views should only be defined when Template Creators have *a priori* content that should be available in any web-site following the modeled Template.

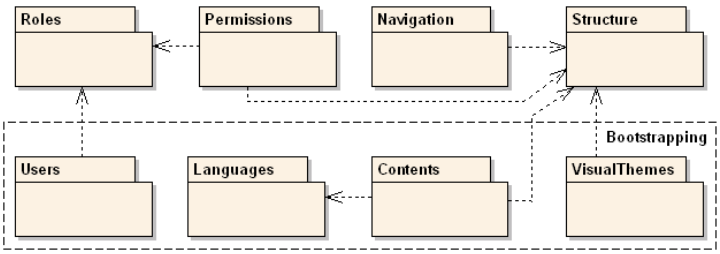


Fig. 3: The views involved in the definition of a Web-Site Template.

The Structure view is the most important, as it conveys the web-site's page structure by using a set of CMS-oriented concepts: (1) *WebSite*, which represents the web-site itself and serves both as a container for Dynamic WebPages and as the element that will import Toolkits (explained further down this text); (2) *Dynamic WebPage*, representing the dynamically-generated pages (in the sense that their contents can be changed through the CMS interface) that users will access; (3) *Container*, which is modeled within a specific area of a Dynamic WebPage and holds a set of WebComponents; and (4) *WebComponent*, representing the "units of functionality" (e.g., Blog, Forum) with which the user will interact. The Structure view is further divided into two smaller views, the *Macro Structure view* and the *Micro Structure view*. The former specifies a "bird's eye" view of the web-site, modeling only the existence of Dynamic WebPages and the relationships between them, while the latter is where each Dynamic WebPage's

structure is specified (i.e., what WebComponents are in the Dynamic WebPage, their location, and their order relative to each other). Figure 4 presents the abstract syntax for the Structure view, where the previously-mentioned concepts can be observed.

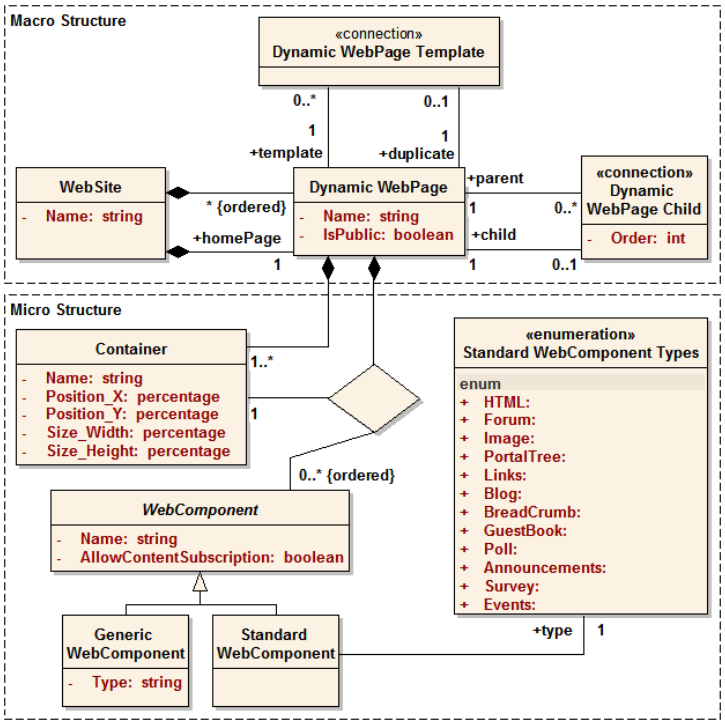
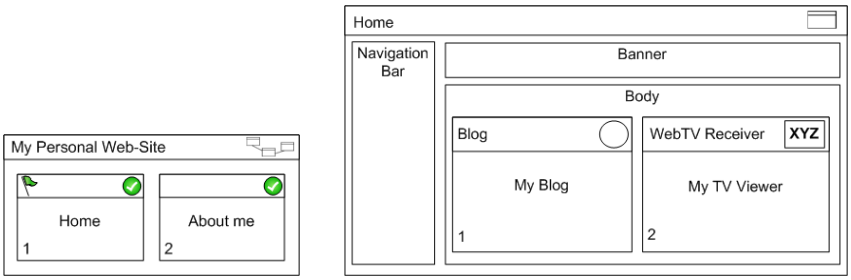


Fig. 4: The abstract syntax for the Web-Site Template's Structure view.

On the other hand, Figure 5 depicts two examples of the Structure view's concrete syntax: Figure 5a illustrates the Macro-Structure view, namely a simple Web-Site containing only two Dynamic WebPages, while Figure 5b shows the definition of a Dynamic WebPage (including Containers and WebComponents) in the Micro-Structure view. The concrete syntax of CMS-ML was defined with the purpose of being easy to understand and to draw manually, without requiring that specialized modeling tools be used in order to create CMS-ML models.

The behavior aspect is only specified in Toolkits (described next) because (1) behavior is usually defined by the WebComponents available in the CMS (e.g., an HTML WebComponent will behave differently than a Forum WebComponent), and (2) even CMS administrators are typically unable to change the web-site's behavior itself, and can only change some parameters regarding specific behavior of the CMS.



(a) Macro-Structure view: Web-Site and Dynamic WebPages.

(b) Micro-Structure view: Dynamic WebPage with Containers and WebComponents.

Fig. 5: The concrete syntax for the Web-Site Template's Structure view.

Due to text size constraints, the abstract and concrete syntaxes of CMS-ML will not be presented in greater detail in this paper, although they will be made available in the very near future at our research group's web-site[1].

3.3 Toolkit Modeling

A Toolkit can be regarded as a "task-oriented extension of CMS elements", as it enables the addition of new CMS-related concepts (namely Roles and WebComponents) oriented towards supporting a particular set of tasks and the corresponding domain model. Like a Web-Site Template, a Toolkit is defined according to a set of views (shown in Figure 6): (1) the *Tasks view*, which deals with the user tasks that the Toolkit should support; (2) the *Roles view*, specifying the Roles that are to perform those tasks; (3) the *Domain view*, which specifies the domain model that is subjacent to the Toolkit's tasks; (4) the *States view*, dealing with the lifecycle of the entities that the tasks are to manipulate; (5) the *WebComponents view*, specifying the WebComponents that will support the tasks; (6) the *Task Interface view*, which establishes mappings between Roles, Tasks and WebComponents, and determines which Roles can do what actions with each of the Toolkit's WebComponents; and (7) the *Side-Effects view*, which establishes side-effects that the modeled Tasks and WebComponents will have.

The Tasks, Roles and WebComponents views are the most important in a Toolkit. The Tasks view allows the Architect to define user tasks as orchestrations of *Steps* which may involve user interaction (similarly to UML's Activity diagrams). The Roles view (not directly related to CMS Roles) models the different kinds of behavior – *Roles* – that the web-application should expect. Finally, the WebComponents view is where the Toolkit's UI (*WebComponents* and *Support Pages*) is specified using *WebElements*, by creating complex UI structures from simpler ones; in turn, WebElements are further divided into *Simple WebElements* (e.g., button, image), *WebContainers* (e.g., DIVs, pop-ups) and *HTML*

[1] `http://isg.inesc-id.pt`

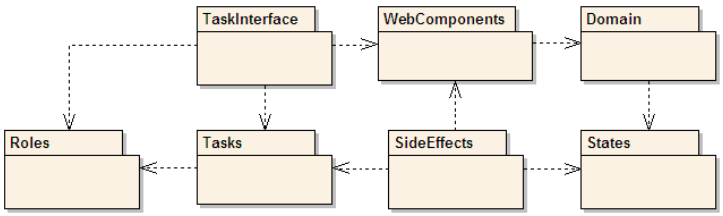


Fig. 6: The views involved in the definition of a Toolkit.

Elements (for cases in which Simple WebElements are not sufficiently adequate). This variety of web interface elements allows the modeling of relatively complex web interfaces using CMS-ML.

3.4 Importing Toolkits

Toolkits can be used in Web-Site Templates or even in other Toolkits, by means of the "Toolkit Import" modeling element, a relationship between a Toolkit (the "imported" element) and either a Web-Site Template or a Toolkit (the "importer"). This relationship is transitive, which means that importing a Toolkit T_1 will automatically import all Toolkits that have been imported by T_1. Also, it is possible to import more than one Toolkit into a Web-Site Template or Toolkit, enabling the composition of Toolkit functionalities in a simple manner.

When importing a Toolkit into a Template, the elements defined in the Toolkit's Roles and WebComponents views become available as new Template modeling elements. When importing a Toolkit into another Toolkit, the elements in the imported Toolkit's Tasks, Domain and States views (but not the Roles or WebComponents views) can be used or specialized by the importer.

It is very important to highlight that Web-Site Templates and Toolkits are located in *different conceptual levels*. While Web-Site Templates are meant to create abstractions of concrete web-sites (i.e., models of those web-sites) by using CMS-oriented elements, Toolkits use generic modeling elements (e.g., *Entity*, *Task*) to create new CMS-oriented modeling elements (namely Roles and WebComponents). Because instances of Toolkit *Role* and *WebComponent* are also automatically considered as specializations of the Web-Site Template's *Role* and *WebComponent* concepts (much like *Generic WebComponent* and *Standard WebComponent* are specializations of *WebComponent*, as the reader can see in Figure 4), Template Creators can then use those Toolkit Roles and WebComponents to create Web-Site Templates exactly in the same manner as when using the pre-defined Template modeling elements.

Figure 7 depicts the metamodel levels that are considered by CMS-ML: the "Toolkit", "Web-Site Template" and "Web-Site Instance" models are created (and changed) by designers, while the "Task Modeling", "Domain Modeling" and "CMS" models are fixed and cannot be changed by designers. In level ML3, Toolkit designers can create instances of generic modeling elements (from Task

Modeling and Domain Modeling, located in ML4) in order to define new elements (Roles and WebComponents) that specialize CMS modeling elements. In level ML2, Template Creators can then use the CMS modeling elements, as well as other modeling elements created in Toolkits, to define a Web-Site Template. In level ML1, the Web-Site Template will be used to create an instance model for a particular CMS installation; this instance model, in turn, will be representing concrete entities that are located in ML0 (the "reality" level, so to speak). Note that the metamodel layers from ML2 to ML0 are actually very similar to what can be found in the OMG's specification of UML [7], because their purpose is nearly the same.

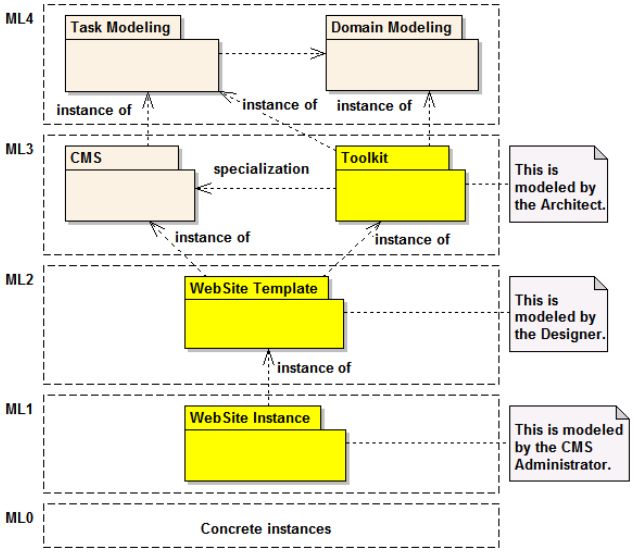


Fig. 7: The metamodel levels considered by CMS-ML.

The rationale for this metamodel level design is to: (1) address language extension in a simple, yet elegant, manner; (2) reduce the accidental complexity [8] that is usually derived from using "type–instance"-like modeling patterns in the same modeling level; and (3) obey the "strict metamodeling" doctrine [9], which states that it should be possible to fully understand any metamodel level as being instantiated from only the metamodel level immediately above it (a consequence of this is that there should be no "instance-of" relationships crossing more than one metamodel-level boundaries).

4 Discussion and Related Work

Besides the CMS-ML presentation done in Section 3, the language does pose some aspects that deserve further discussion. In this section we present and

discuss those aspects, while relating them to some external work that we consider relevant for our research, namely the *Web Modeling Language* (WebML) [10,11] and *UML-based Web Engineering* (UWE) [12,13].

One of the first aspects to discuss is the reason why CMS-ML was defined without using any particular meta-metamodel, as opposed to using a mechanism such as a UML Profile [7] (in the same manner that UWE was defined). This is made even more relevant by the fact that, to our knowledge, UML's modeling elements do not present any semantics that contradict the semantics of CMS-ML. However, it would be problematic to represent the Toolkit aspect as a UML Profile, in such a way that elements defined in a Toolkit could then be used to define the Web-Site Template (another UML Profile). This problem is due to the fact that UML (and UML-oriented tools) does not explicitly consider *metamodeling* as an important issue [14], which in turn usually leads to a much greater degree of accidental complexity [8] (i.e., making modeling languages more complex than necessary). It should also be noted that other web-engineering modeling languages do not consider their extension – in the sense of adding new modeling elements – as an important concern (although WebML does define some generic Data-Units, which must be later implemented in source-code, to cover cases in which it is not expressive enough).

At first sight, the Web-Site Template may appear to be adequate for modeling page-centric CMS web-sites (e.g., WebComfort [15], DotNetNuke [16]), but not content-centric CMS web-sites (such as Joomla [17] or Drupal [18]). However, this Template ultimately reflects how *users* will see the web-site, instead of reflecting the concepts that the CMS itself is using. Considering that even web-sites using content-centric CMS systems have a certain structure perceived by their users, we believe that CMS-ML can adequately model web-sites based on content-centric CMS systems.

Another aspect to discuss is how CMS-ML deals with the possible semantic gap between the Toolkit's WebComponents (i.e., UI) and Domain views. While UWE sometimes requires that its Content Model be "tweaked" to particular details of other Models (namely the Presentation Model) [19], WebML defines the Derivation Model to define a layer that establishes mappings between the Data Model and other WebML Models (namely the Hypertext Model). CMS-ML Toolkits also do not require the Domain view to be oriented towards the needs of other views, because the WebComponents view contains a set of modeling elements that allow Architects to specify what parts of the Domain view are to be displayed or used, by using a "binding context" mechanism inspired by our previous work in XIS2 [20].

It is also important to discuss the language's expressiveness and its adequacy to model real-world web-applications. CMS-ML is not very expressive when compared to other languages such as WebML or UWE. However, this level of expressiveness is intentional: because CMS-ML is a part of a larger approach – involving a *set of languages* – the rationale was to reduce the number of modeling elements in this language, in order to make it easier to learn. Nevertheless, unaddressed requirements cannot be just "ignored": that is why CMS-ML de-

fines the concepts of *Unaddressed CMS Requirement* and *Unaddressed Toolkit Requirement*, which are just textual segments (similar to UML comments or constraints) that can be associated with any CMS-ML modeling element. These concepts bring added value to the model (and are not just decorative), because they will be translated to "reminders" in the corresponding CMS-IL models.

The final aspect to highlight is the fact that we believe accidental complexity [8] in CMS-ML has been reduced to a minimum. This is mainly due to the fact that the CMS and Toolkit modeling elements do not include the means to establish "instance-of" relationships between elements; this kind of relationship would become necessary to create models using the "type–instance" modeling pattern, which in turn is usually a source of accidental complexity.

5 Conclusions and Future Work

In this paper we have introduced CMS-ML, a graphical language for the high-level modeling of CMS-based web-applications, aimed at allowing non-technical users to easily understand and change a web-site's model. To achieve its goal, CMS-ML defines a set of CMS-oriented views and can be extended with new concepts. This language is a part of a larger approach for the development of this kind of applications, which explains its lack of expressiveness to deal with concrete implementation details, such as algorithm specification.

Regarding future work for CMS-ML, there are still some open issues, of which we highlight here the ones that we consider most important for the time being.

One of those issues is expressiveness. The CMS-ML language is the result of a tradeoff between language complexity, expressiveness, and how often a given pattern can be found in existing web-applications. However, we acknowledge that this tradeoff will always have a certain amount of subjectivity to it. We consider it necessary to try and minimize this subjectivity factor, in order to make the language more practical, adequate and useful for real-world scenarios. Furthermore, the fact that CMS-ML is independent of any particular CMS makes it unable to use CMS-specific concepts (e.g., Workflow), a problem that we wish to address in the future (likely by using an approach similar to what we did with the Toolkit—Web-Site Template metamodels).

The other issue, closely related to the expressiveness issue, is the validation of CMS-ML. To minimize the subjectivity factor and validate the language in case-studies, we intend to use it for modeling sites and applications with a reasonable degree of complexity. Although we are already validating the language in some academic case-studies, we will also use it in some more complex real-world scenarios, such as WebC-Docs [21], a document management system that we have developed in the context of our research regarding CMS-based web-applications.

References

1. Boiko, B.: Content Management Bible. John Wiley & Sons, Hoboken, New Jersey, U.S.A. (December 2001)

2. The CMS Matrix. Retrieved May 31, 2010 from http://www.cmsmatrix.org
3. Carmo, J.L.V.d.: Web Content Management Systems: Experiences and Evaluations with the WebComfort Framework. Master's thesis, Instituto Superior Técnico, Portugal (December 2006)
4. Saraiva, J.d.S., Silva, A.R.d.: The WebComfort Framework: An Extensible Platform for the Development of Web Applications. In IEEE Computer Society, ed.: Proceedings of the 34th EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2008). (September 2008) 19–26
5. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer **39**(2) (February 2006) 25–31
6. Saraiva, J.d.S., Silva, A.R.d.: CMS-based Web-Application Development Using Model-Driven Languages. In IEEE Computer Society, ed.: Proceedings of the Fourth International Conference on Software Engineering Advances (ICSEA 2009). (September 2009) 21–26
7. OMG: Object Management Group – Unified Modeling Language: Superstructure – Specification Version 2.0 (August 2005) Retrieved May 31, 2010 from http://www.omg.org/spec/UML/2.0/Superstructure/PDF/.
8. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software and Systems Modeling **7**(3) (July 2008) 345–359
9. Kühne, T.: Contrasting Classification with Generalisation. In Kirchberg, M., Link, S., eds.: Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM 2009). Volume 96 of CRPIT., Australian Computer Society (January 2009) 71–78
10. WebML.org. Retrieved May 31, 2010 from http://www.webml.org
11. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann (2003)
12. UWE – UML-based Web Engineering. Retrieved May 31, 2010 from http://uwe.pst.ifi.lmu.de
13. Kroiß, C., Koch, N.: UWE Metamodel and Profile: User Guide and Reference. Technical Report 0802, Ludwig-Maximilians-Universität (February 2008) Retrieved May 31, 2010 from http://uwe.pst.ifi.lmu.de/download/UWE-Metamodel-Reference.pdf.
14. Saraiva, J.d.S., Silva, A.R.d.: Evaluation of MDE Tools from a Metamodeling Perspective. Journal of Database Management **19**(4) (October/December 2008) 21–46
15. SIQuant: WebComfort.org. Retrieved May 31, 2010 from http://www.webcomfort.org
16. DotNetNuke. Retrieved May 31, 2010 from http://www.dotnetnuke.com
17. Joomla CMS. Retrieved May 31, 2010 from http://www.joomla.org
18. Drupal CMS. Retrieved May 31, 2010 from http://drupal.org
19. UWE – Tutorial. Retrieved December 9, 2009 from http://uwe.pst.ifi.lmu.de/teachingTutorial.html
20. Silva, A.R.d., Saraiva, J.d.S., Silva, R., Martins, C.: XIS – UML Profile for eXtreme Modeling Interactive Systems. In: Fourth International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007), Los Alamitos, CA, USA, IEEE Computer Society (March 2007) 55–66
21. SIQuant: WebComfort.org – WebC-Docs. Retrieved May 31, 2010 from http://www.webcomfort.org/WebCDocs

Enterprise Governance and DEMO

Guiding enterprise design and operation by addressing DEMO's competence, authority and responsibility notions

Miguel Henriques, José Tribolet, and Jan Hoogervorst

Department of Information Systems, Instituto Superior Técnico (IST-UTL), Lisboa
miguel.henriques@ist.utl.pt,jose.tribolet@inesc.pt,jan.hoogervorst@sogeti.nl

Abstract. The lack of an organizational competence that embodies the capacity to restrict the enterprise undesirable design freedom and guide the subsequent operation, from a holistic point of view, leads to incoherence and inconsistency among the enterprise elements. The research brings forward the importance of this organizational competence labeled enterprise governance (EG) in defining DEMO's ontological models and using their subsequent authority, responsibility and competence notions to guide the enterprise dynamics. Based on these results, the article provides a reference method for the EG to define a set of normative outputs, derived from these three notions addressed in the enterprise ontologic models, comprising a set of principles to address enterprise integration and a set of rules to deal with on-going organizational changes while addressing security issues.

Key words: Enterprise Governance, Enterprise Design, Enterprise Architecture, Enterprise Ontology, Enterprise Engineering

1 Introduction

In a world of growing business dynamics, high rates of technological and organizational changes, enterprises need to be continuously (re)designed and (re)engineered in order to achieve strategic and operational success. Our research will be built around the enterprise development theory within the enterprise engineering discipline.

In this context, we address one core problem: the lack of coherence and consistency among the various enterprise elements resulting from the enterprise incapacity to effectively build the enterprise strategy into design and manage the subsequent changes at the operational plane from a holistic point of view [11, 1]. It is estimated that between 70% and 90% of strategic initiatives tend to fail [10, 3]. Researchers argue that such failures in most cases result from inadequate strategy implementation in the sense that if the enterprise aspects are not addressed in design by thinking about the enterprise as an organic whole the enterprise will not be able to operate as a unified system and the strategy implementation will tend to fail [1].

We argue that there are two interrelated main reasons behind this problem. First, the enterprise does not have the ability to apply in practice the design theory from the enterprise engineering discipline and thus, it is unable to master the enterprise complexity and to develop an integrated enterprise system [3]. Second, absence of an organizational competence that should guide globally the enterprise development process and subsequent changes in order to ensure the correct use of this theory [7].

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 473–476

2 An organismic governance approach

The systemic approach to problems focuses on systems taken as a whole, instead of their parts taken individually [1, 11]. Within this view an enterprise is perceived as a goal-seeking system, intentionally designed by a set of interacting human beings behaving according to assigned authority and corresponding responsibility against a common background of social norms and values. Mastering the enterprise complexity and guide the design[1] in a holistic manner is essential during the implementation of the strategy to achieve integration among enterprise components [13]. However, this notion of guidance is often associated with a 'mechanistic' perspective (top-down, management and control oriented). Within this perspective, there is no adequate competence and attention for addressing the enterprise design (we refer to [9, 11]).

Enterprise strategy and design subjects transcend the capabilities of the corporate governance and IT governance disciplines and can only be addressed within the overall scope of EG (we refer to [6]). The EG consists in an integrated whole of knowledge, skills and technology, whereby employees are viewed as the crucial core for continuously exercising guiding authority over enterprise strategy and *architecture* development, and the subsequent design, implementation and operation.

Architecture notion has been associated to a descriptive approach perceived as a "blueprint" of the system construction and a prescriptive approach concerning design guidance [5, 8, 12]. A prescriptive approach of the concept must be exercised comprising "consistent and coherent set of design principles" defined by the EG in order to guide the design by restricting its undesirable freedom [3].

3 Ontology - DEMO methodology

The models resulting from the design process approach the system construction at different levels of abstraction. At the "highest level" there is the ontological model and at the "lowest level" there is the implementation model [3]. The core meaning of system ontology in our thesis context is a model for describing and understanding the construction and operation fully independent of the way the system is implemented which is coherent, comprehensive, consistent and concise (we refer the reader to [2]).

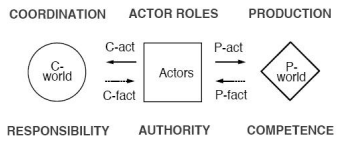


Fig. 1: The construction axiom[2]

Competence	"The ability of a subject to perform a particular type of P-act as well as the corresponding C-acts."
Authority	"The condition of being allowed to act." It can be assigned through authorization and delegation.
Responsibility	"The quality of a subject to be committed to the coordination facts he or she has performed, as well as to coordination acts that are addressed to him or her."

Fig. 2: Authority, responsibility and competence[2]

The scientific root of DEMO is the Ψ-theory, we outline its four essential axioms according to [2]. *The construction axiom* indicates that an enterprise consists of actors performing productions acts (P-acts) to bring about the enterprise mission and coordination acts (C-acts) to enter into and to comply with commitments. *The operation axiom* says that for every type of C-act there is an action rule to guide enterprise actors. *The transaction axiom* argues that P-acts and C-acts occur in generic recurrent patterns performed by two actors called transactions. *The abstraction axiom* distinguishes three human abilities to perform C-acts: forma, informa and performa.

[1] Design is percieved as the production process of conceptual models of a system [3].

4 Proposed model and underlying reference method

The conceptual model that relates the notions of EA, ontological models and EG is outlined in figure 3. The EG uses the ontological models to master in a holistic manner the enterprise complexity and devise a set of outputs that will guide the enterprise design as well as the enterprise execution plan (operation). Governance outputs can be divided in (1) principles devised from all the enterprise design domains (traceable with the enterprise areas of concern) that will restrict the design process (EA notion), and (2) outputs retrieved from DEMO models based on the notions of competence, authority and responsibility, which will guide the detailed design and the enterprise operation. Principles purpose is to deal with the article main concern (the lack of integration among the enterprise elements at design level), while rules will ensure that the enterprise operation conforms the enterprise design.

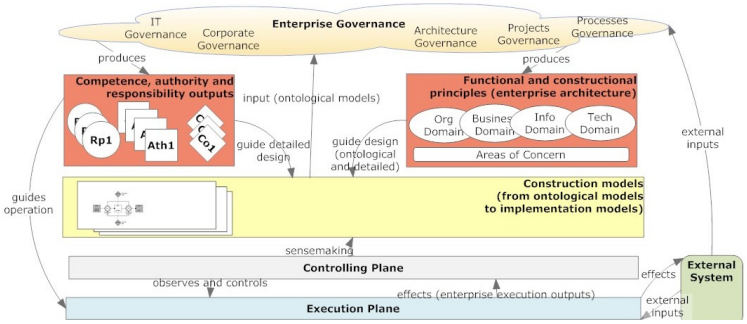


Fig. 3: Enterprise Governance, Enterprise Ontology and Enterprise Architecture

Based on DEMO's theory and governance themes discussed above, as well as researches in the field of responsibility and security modeling [4], we infer a reference method to govern the enterprise dynamics with DEMO depicted in table 1.

Method stage	Observations
(1) identify the enterprise actor roles	The task of identifying actor roles is already provided by DEMO: they can be retrieved from the enterprise construction (interaction or interstriction) models.
(2) identify the areas of responsibility	For one specific transaction, the Process Model defines all the C-acts that an actor role is allowed to perform. Hence, the responsibility areas are rigorously defined in DEMO's Process Models
(3) identify the competence domains and define a set of competence principles for each actor role	Competence domains can be perceived as attributes that will guide the evaluation process to check if a person has the adequate competence to exercise its job. Competence principle purposes to restrict the detailed design freedom regarding the actors production acts.
(4) define all the authority rules for each actor role (who has the right to exercise authorization and delegation and in which conditions)	For this purpose should be defined (1) the acts that each actor need to do, (2) who is allowed to access what information and the information that must be audited for each actor role. The illustration of these three requirements can be further transformed in eligible authorization rules. Consequently, if the acts and information required do not exist, the actors are not allowed to do and see anything else than what is specified. In this fashion, DEMO enforces a role-based access control.
(5) define the action/coordination rules for each type of C-/P-act	operational rules consist of two categories: the coordination rules for guiding the coordination activities (responsibility) and the production rules for guiding production activities (competence).
(6) identify responsibilities pre-conditions and post-conditions needed to discharge a responsibility	Since the c-acts are represented as action rules, then we can assume that there is a set of pre-conditions that must hold before an action rule can start. After an instance of discharging a responsibility there are statements about the environment and the agent that are true, these are the post-conditions (formal statements).
(7) create a list of exception conditions	Exception conditions list express all the exceptions that need to be handled when occurs a deviation regarding the enterprise norms. In this fashion, when an exception is detected the adequate mechanisms and actors will be properly alerted

Table 1: Proposed reference method

The reference model and method are being validated in the Portuguese Justice System, in particular at DIAP. The benefits of these artifacts are being demonstrated as essential to identify and deal with the inconsistent and incoherent requirements by devising principles from a global architecture framework to support the operation of all the justice procedures and actors (e.g. the lack of coordination between internal and external entities in crime investigations), and to deal with issues such as security, information access, traceability of the agents' actions, among others, which are addressed at the design level and its correct execution ensured at the operational level in order to deal with the continuous organizational changes.

5 Conclusion

This article described the potential of bringing together the notions of enterprise governance and enterprise ontology (within DEMO). On one hand, the EG should be associated to an organismic perspective responsible for guiding the enterprise strategy and enterprise development by restricting the undesirable design freedom in the form of principles (architecture notion) and guiding the subsequent enterprise operation in the form of operational rules. On the other hand, DEMO provides a methodology to represent the enterprise essence in an intellectual manageable way.

Based on the research in this field of knowledge, we developed a conceptual model and an underlying method to support the EG in defining a set of normative outputs. This method uses the notions of competence, responsibility and authority within DEMO to deal with the continuous changes and restrict the detailed design process (competence principles regarding P-acts), to deal with security issues associated to information access and responsibility transfer (authority rules), and to help identifying requirements that are inconsistent and incoherent and mutually align enterprise design and operation (production and coordination rules, exceptions list).

References

1. Russell L. Ackoff. *Ackoff's Best: His Classic Writings on Management.* Wiley, 1999.
2. Jan L. G. Dietz. *Enterprise Ontology: Theory and Methodology.* Springer, 2006.
3. Jan L. G. Dietz. *Architecture - Building strategy into design.* Academic Service, 2008.
4. John Dobson Guy Dewsbury. *Responsibility and dependable systems.* Springer, 2007.
5. Van Haren. *TOGAFTM The Open Group Architecture Framework A Management Guide.* Van Haren Publishing, 2005.
6. Jan A. P. Hoogervorst. *Enterprise Governance and Enterprise Engineering (The Enterprise Engineering Series).* Springer, 1 edition, February 2009.
7. Jan A. P. Hoogervorst and Jan L. G. Dietz. Enterprise architecture in enterprise engineering. *Enterprise Modelling and Information Systems Architectures*, 2008.
8. IEEE1471. Recommended practice for architectural description for software- intensive systems. Technical report, IEEE Computer Society, 2000.
9. Michael C. Jackson. *Systems Thinking.* Wiley, 2003.
10. Robert S. Kaplan and David P. Norton. *Strategy maps.* Harvard Business School Press, Boston, Mass., 2004.
11. Eberhardt Rechtin. *Systems Architecting of Organizations: Why Eagles Can't Swim.* CRC, 1 edition, July 1999.
12. Pedro Sousa, Artur Caetano, Andre Vasconcelos, Carla Pereira, and Jose Tribolet. *Enterprise architecture modeling with the Unified Modeling Language.* Idea Group Publishing, 2006.
13. Gerald M. Weinberg. *An Introduction to General Systems Thinking.* Dorset House Publishing Company, Incorporated, April 2001.

Especificação, Verificação e Teste de Sistemas Críticos

A (Very) Short Introduction to SPARK: Language, Toolset, Projects, Formal Methods & Certification

Eduardo Brito

CCTC / Departamento de Informática
Universidade do Minho
Campus de Gualtar, 4710-057 Braga, Portugal
edbrito@di.uminho.pt

Abstract. Guidelines for the development of software in safety-critical systems usually restrict programming languages, removing features that are unsafe and/or hard to thoroughly test and certify. There are also recommendations and demands in newer guidelines for the use of formal methods, as a way to achieve high assurance software. SPARK is a strict subset of Ada that was designed to have unambiguous semantics and that aimed at formal verification from the start. In this paper we present the SPARK language, its toolset, examples of projects where it has been used and argue why SPARK is relevant for academia and industry, especially for people interested in formal verification and safety critical systems. We also point directions for an improved use of SPARK. Concurrency will not be addressed in this paper.

Resumo: Os parâmetros para o desenvolvimento de *software* em sistemas *safety-critical* normalmente restringem as linguagens de programação, removendo características inseguras e/ou difíceis de testar e certificar de forma rigorosa. Existem também recomendações e exigências em novos parâmetros para o uso de métodos formais como forma de obter software com mais garantias. Neste artigo apresentamos a linguagem SPARK, o seu conjunto de ferramentas, exemplos de projectos onde foi usada e argumentamos o porquê do SPARK ser relevante para a academia e a indústria, especialmente para pessoas interessadas em verificação formal e sistemas *safety-critical*. Apontamos também direcções para um uso melhorado do SPARK. Neste artigo não abordaremos concorrência.

1 Introduction

Safety-critical systems are some of the most demanding systems when it comes to certification. This certification is not only required from the software but from the system as a whole, although in this paper we will only address features that are directly related to software.

When developing software for these systems, one of the most used programming languages is Ada[34]. Ada is a well known and respected language used in several domains of safety-critical software development. Furthermore, Ada is

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 479–490

also a general purpose programming language, which has been kept up-to-date with several major revisions (83, 95, 2005 and now the future 2012) which have been standardized and thoroughly documented in the Ada Reference Manual[1].

When developing safety-critical systems, the language features that are allowed to be used are rather restricted. This is enforced by several guidelines (which vary depending on the application domain) with the aim of having software that is easier or, in some cases, possible, to certify mainly through thorough testing within a reasonable amount of time/cost/effort. These restrictions also depend on the integrity level (e.g. SIL, ASIL, DO-178B level) aimed by the software component that is being developed. This integrity level has to be given not only by the importance it has on the system but how it can affect other systems.

Some guidelines now incorporate recommendations and/or demand the use of formal methods so that it can be mathematically justifiable that the software behaves as intended, thus producing software that provides a higher assurance.

SPARK[6] is on the convergence of these aims. It is a strict subset of Ada that was designed to have unambiguous semantics and that aimed at formal verification from the start. It removes several features of the Ada language while still retaining enough expressive power so that it can be used in realistic scenarios.

The rest of this paper is organized as follows: Sec. 2 overviews design-by-contract and behavioral interface specification languages, two approaches that are at the basis of SPARK; in Sec. 3 we briefly present the SPARK language and its toolset; Sec. 4 surveys projects, both in industry and academia, where SPARK is/was used; in Sec. 5 we point out various aspects that could enhance SPARK and further justify the use of SPARK as a framework for rigorous software development; Sec. 6 concludes the paper.

2 Contracts & Specification

In this section we present what is Design-by-Contract (DbC) and in what ways it resembles and differs from Behavioral Interface Specification Languages (BISL). It is important to describe this since the language SPARK which we will be focusing on (as well as several others that aim at source code verification) has its approach rooted in this.

2.1 Design by Contract™

The term "Design by Contract" was first coined by Bertrand Meyer[26] and is largely associated with the Eiffel[27] programming language, an Object-Oriented programming (OOP) language, as part of the language's philosophy and design process. The term is also a registered trademark and some authors prefer to use the expression Programming by Contract.

The main ideas behind DbC are: a) to document the interface of modules[2] and its expected behaviour, b) to help with testing and debugging and c) to

[1] Usually referred to as ARM.

[2] The term modules is equivalent to package or class.

assign blame when a contract is breached. This is achieved by having structured assertions such as invariants and pre- and post-conditions.

In DbC, following the tradition of Eiffel, assertions are boolean expressions, often using subprograms[3] written in the same language as the host programming language and are intended to be checked at runtime (*runtime assertion checking (RAC)*), by executing them. Writing assertions in this way is friendlier to developers but it makes formal verification impossible because contracts are also code and not mathematical specifications describing the properties to be ensured.

In article[19] it was illustrated how Eiffel could have helped prevent the bug in the software of Arianne V, thus avoiding one of the most expensive software errors ever documented. What is stated is that the error that made Arianne V go wrong could have been avoided if the pre-conditions for the subprogram that failed had been clearly stated in the code. If the dependencies were clearly documented in the code then the verification & validation team would have been aware of what could (and did) generate a runtime error.

To sum it up, DbC is used to document source code and to have the program checked while it is executing, using structured annotations that are written as boolean expressions of the host programming language.

2.2 Behavioral Interface Specification Languages

Behavioral Interface Specification Languages was a term introduced with Larch [15]. The Larch family of languages had the specification, in a mathematical notation, written alongside the source code (Larch supported C, C++, Smalltalk and Modula-3). This resembles DbC but it differs greatly from traditional specification languages (SL), such as Z[32] and VDM[20], where the specification is written as a separate entity with no relation to the implementation.

Larch was focused mainly on specifying, with brevity and clarity, the interface of subprograms and datatype[4] invariants. Although some specifications[5] were executable, executability of specifications was not an objective of the Larch family of languages; this is the exact opposite of DbC. These specifications, along with the source code, would give rise to proof obligations.

The way that specifications are written in BISLs and SLs are similar. In both approaches, the specifications are written using a well-defined formal notation that is related to a well-defined formal logic. These formal definitions do not use expressions from the host language although they may look similar in some cases. This is another difference regarding DbC.

It is possible to animate/execute specifications written in some SLs (depending on the available tools) but this is still different from DbC and BISL because

[3] Subprograms is being used as a more generic way to refer to methods, functions, procedures and subroutines.

[4] Datatypes can also refer to classes. The term in LCL (Larch for the C language) was used to refer to structures.

[5] When talking about BISL, the terms “annotation” and “specification” are mostly interchangeable

we are animating a specification and not an implementation. Even so, we could argue that it is possible to refine a specification into an implementation.

While writing annotations in a mathematical notation is very expressive and particularly helpful for program verification, Larch has showed us that excessive mathematical notation can lead to the poor adoption of a BISL. JML[22] is a modern example of a BISL, rooted on the principles of Larch, which has taken this into account. JML avoids excessive mathematical notation, while having a mathematical background, and has gained several supporters in the academic and industrial arena, especially with the success of JavaCard[33] verification tools[3,24].

JML, as noted in[23,11], is associated with a set of tools that makes possible to overcome the typical non-executable nature of BISLs. Also, besides being able to do RAC, it also allows for the formal verification of Java programs, given the right tools/frameworks.

ACSL[6][9] is another interesting and modern BISL. While it has a large influence from JML, it has greater expressive power regarding the definition of mathematical entities.

It is also worth mentioning that BISLs, contrary to the classical notion of DbC, are not only useful for OOP but they can also be applied to other programming paradigms. Larch, for example, had support for C and ACSL was designed specifically for C. It is only because it is more natural to use this type of specifications in the OOP context that it has been largely associated with it.

3 SPARK

In this section we describe the features of the SPARK programming language and how it is supposed to work with the help of its toolset. We will focus mainly on program verification and proof support for SPARK, with the tools that are available from the official toolset distributed by AdaCore and Altran Praxis. This section assumes SPARK GPL 2009 (version 8.1.x), unless it is stated otherwise[7].

3.1 The Programming Language

SPARK stands for SPADE Ada Kernel[8]. SPADE was a previous project from Program Verification Limited, with the same aims as SPARK, but using the Pascal programming language. The company later became Praxis High Integrity Systems and it is now called Altran Praxis.

SPARK is both a strict subset of the Ada language, augmented with source code annotations, and also a toolset that supports its methodology. The SPARK annotations can be thought of as a BISL.

[6] ANSI/ISO C Specification Language.

[7] There are plans to launch a SPARK GPL 2010 version with features from SPARK Pro 9 during this summer[25].

[8] The R in SPARK is only there for aesthetic purposes.

It is a strict/true subset of the Ada language because every valid SPARK program is a valid Ada program; in fact, SPARK does not have a compiler and depends on Ada compilers to generate binary code. SPARK was also cleverly engineered so that the semantics of SPARK programs do not depend on decisions made by a specific compiler implementation (e.g. whether a compiler chooses to pass parameters of subprograms by value or by reference). Although outdated, SPARK also has a formal semantics[28] that is defined using operational semantics and Z notation.

SPARK removes some features of the Ada language such as recursion, dynamic memory allocation, access types[9], dynamic dispatching and generics. It also imposes certain restrictions on the constructs of the language (e.g. array dimensions are always defined by previously declared type(s) or subtype(s)[10]).

On top of the language restrictions, it adds annotations that allow for the specification of data-flow, creation of abstract functions and abstract datatypes and to the use of structured assertions (loop invariants are available but package[11] invariants are not). There is also a clear distinction between procedures (which may have side-effects) and functions (which are pure/free of side-effects and also have to return a value). SPARK provides no way to execute annotations.

Ada has the notions of package (specification) and package body (implementation); these are the building blocks for OOP in SPARK and Ada, although we may choose not to use the OOP features of the language. SPARK restricts OOP features that make the code hard to verify, such as dynamic dispatching. This issue is also addressed in[7], relatively to Ada 2005.

Package specifications can have abstract datatypes and abstract functions, which can then be used to define an abstract state machine. When coding the implementation, the abstract datatypes have to be refined into concrete datatypes, using the own annotation, and the definitions for abstract functions are written into *proof rule* files. These files are used when trying to discharge verification conditions.

Although it is not our aim to talk about the information flow capabilities of SPARK, it is interesting to note that in SPARK Pro 9, which has the SPARK 2005 version of the language, is also possible to validate secure and safe flow of information between different integrity levels and information security levels (i.e. unclassified, top secret).

All these features enable the possibility of developing software using the Correctness by Construction approach[12] where a software component is designed and developed with the aim of being formally verified.

[9] Users not familiar with Ada should note that these are equivalent to pointers.

[10] In Ada, the type mechanism allows for the definition of *ranges*. These ranges are limited by a lower and upper bound, known as T'First and T'Last where T is the type. It is also possible to get the range using T'Range.

[11] As previously stated, packages are equivalent to modules and classes

[12] This is different from the refinement approach taken by the B Method.

Finally, it should be noted that SPARK has support for a subset of Ravenscar[13] dubbed RavenSPARK[30].

3.2 Toolset & Program Verification

The SPARK toolset is what enables the use of SPARK. By not having a compiler, it must have a tool that checks the restrictions of the SPARK language; this tool is called the Examiner. The Examiner is also the verification condition (VC) generator (VCGen).

With the SPARK toolset we can use its proof tools to discharge the VCs. The Simplifier is the automated theorem prover and tries to discharge the VCs by using predicate inference and rewriting. While the tool is very successful in discharging VCs related to safety[17], it is not as capable of discharging VCs related to functional correctness as other modern provers[18]. The Proof Checker is the interactive prover/proof assistant of the toolset. It is a difficult tool to use and there is not much documentation provided and/or available.

An auxiliary tool, that ties all these tools together, is called POGS. POGS stands for Proof ObliGation Summarizer. It generates a report with much information, including warnings and errors related to the code but, most importantly, it constructs the report regarding the VCs that have been proven, that remain to be proven and those that have been shown to be false.

The SPARK Pro 9 toolset also has another tool called the ZombieScope. This tool provides analysis related to dead paths. A dead path is a piece of code that will never be reached; this is most likely caused by programming errors. This tool generates dead path conditions (similar to VCs) and tries to disprove that something is not reachable; if it fails, then it is a dead path. This information also appears on the report generated by POGS, as expected.

The GPS (GNAT Programming Studio), which is provided by AdaCore, has support for SPARK and it has been constantly updated so it is easier to use the SPARK toolset inside an IDE[14] that is familiar to some Ada users. GPS is not integrated in the toolset but it is tightly coupled to it.

4 Projects & Methodologies Related to SPARK

In this section we illustrate the capabilities of SPARK by considering both projects where the language has been used and also some methodologies and tools that use or generate SPARK. As a preview, we will talk about Tokeneer, Echo, SCADE and the C130J helicopter in this section.

4.1 Industrial & Academic Projects

Tokeneer[5] was an industrial project developed under a contract with the NSA, to show that it was possible to achieve Common Criteria EAL5 in a cost effective

[13] Ravenscar is a limited subset of concurrency and real-time of the Ada language.
[14] Integrated Development Environment.

manner. The purpose of the project was to develop the "core" part of the Tokeneer ID Station (TIS). The TIS uses biometric information to restrict/allow physical access to secured regions.

The project was successful, exceeding EAL5 requirements in several areas, and was allowed to be publicly released during 2009, along with all deliverables and a tutorial called "Tokeneer Discovery". In 9939 lines of code (LOC), there were 2 defects; one was found by Rod Chapman, using formal analysis, and another during independent testing, thus achieving 0,2 errors per kLOC. Tokeneer has been proposed as a Grand Challenge of the Verified Software Initiative[36].

Praxis also published a short paper[12] where they described their industrial experience with SPARK (up until 2000). In that paper it is presented SHOLIS (a ship-borne computer system), MULTOS CA (an operating system for smartcards) and the Lockheed C130J (Hercules) helicopter, for which SPARK was used to develop a large part (about 80%) of the software for the Mission Computer.

These case studies illustrate several features of the SPARK language, including how the language eases MC/DC verification by having simpler code, how proofs are maintained and can be used as "regression proofs" instead of "regression testing" and how SPARK can also inhabit multi-language systems. It also shows that SPARK is not appropriate to being adopted late in the development, especially when trying to re-engineer Ada code into SPARK.

Recently[1] SPARK has been chosen as the programming language for a new NASA project, a lunar lander mission. SPARK will be used to develop the software of the CubeSat project. SPARK was also used as a target for a study on verified component-based software, based on a case study developed in SPARK for a missile guidance system[21]. This goes hand-to-hand with the SPARK philosophy of Correctness by Construction.

The Open-DO initiative[15] is also developing a project, called Hi-Lite[16], which uses SPARK in several ways. One of the aims of the project is to create a SPARK profile for the Ada language[17]. While SPARK itself can be viewed as a profile for Ada, the aim of Hi-Lite is to provide a less restrictive SPARK, with the benefits that SPARK has shown over time. Another aim of the project is to write a translator from SPARK (or sparkify) to Why, so that it is possible to use Why's multi-prover VCGen. Why's VCGen has support for interactive provers as well as automated provers. This would provide an alternative toolchain for SPARK besides the one that is maintained by Altran Praxis.

There is also ongoing research work by Paul B. Jackson[18] in translating the VCs generated by the Examiner to SMT-Lib input. This work, that originated from the academia, is scheduled to be included in the SPARK Pro toolset as an experimental feature, in a future release.

[15] http://www.open-do.org

[16] It should be noted that this project is very large and this is just a small portion of the aims of that project.

[17] This is being called sparkify.

4.2 Industrial Tools & Methodologies

The SCADE Suite is a well known software design tool that has been used, for example, by Airbus in the development of the Airbus A340 and is being used in several of the Airbus A380 projects. The SCADE Suite has a certified code generator for SPARK[2]. Perfect Developer[14] is another tool for modelling software for safety-critical systems that has a code generator for SPARK. It is out of the scope of this paper to try and thoroughly explain these tools but, in a very simple way, these tools are driven by model engineering in a formal setting and have been used in large scale projects, especially SCADE.

SPARK is also being used as an essential component of the Echo approach. This approach[37] is able to tightly couple a specification (in PVS[29]) with an implementation. It generates SPARK annotations from the PVS specification; this generated code is then completed to form the implementation. Afterwards, the tool extracts properties from the implementation to check if the implementation conforms to the specification. This approach has been used to formally verify an optimized implementation of the Advanced Encryption Standard.

5 SPARK as a Workbench for Rigorous Software Development

After showing some of the capabilities of SPARK and the support it has from the industry and academia, we now provide some insights on what we believe we can do to improve SPARK by enhancing or adding new features.

5.1 Theoretical Foundations of SPARK & Further Improvements

As it has been stated previously, SPARK has already a formal semantics that is specified with operational semantics and Z notation, but it is already outdated.

We believe that it is important to update the formal semantics of the language, for several reasons. First and foremost, it can be argued that since the domains where the language is used are mostly related to safety-critical and that the language aims at formal verification, it should be formally specified and be kept up-to-date, so that there is no distinction between the formal specification and the implementation.

Also, the semantics is separated into two documents describing the static and dynamic semantics of the language. An unification of the semantics could have benefits (which is also stated in the document but was not carried out). Another question is that the specification, to our knowledge and to what is shown in the documents, was not thoroughly verified (although the use of Z guarantees that at least everything is properly typed). By this we mean that there are no proofs showing relevant properties of the language that are stated, such as expressions being pure (i.e. free of side effects).

Yet importantly, there is only the operational semantics. There is no axiomatic semantics nor weakest pre-condition (or strongest post-condition) calculus nor a specification for a VCGen. While this is implemented in the toolset, an implementation is not a specification (although it could be used as a reference).

For greater formalisation and assurance, we believe that the language would benefit from having a specification for the axiomatic semantics and for the VCGen. The axiomatic semantics could be shown to be sound, regarding the operational semantics and the VCGen could be shown to be sound and correct, regarding the axiomatic semantics. Ideally this would involve mechanical verification using interactive proof assistants. This type of formal verification has already been done by Homeier[16] for a standard While language.

SPARK is ideal for this because it is a real programming language, used in large scale projects, but it is also one of the smallest real languages that do not compromise expressive power, at least for its application domain. Even though it is smaller than most languages, it is much bigger than any toy language.

Thus SPARK provides several challenges for researchers, related to its type system, to the separation between specification and implementation, to the refinement of datatypes and so on and so forth.

There is ongoing work on this area from the author of the paper.

5.2 Adding Features and Tools for Program Verification

One of the most obvious things that is lacking in SPARK is loop variants. While it could be argued that most loop variants in safety-critical systems are trivial, and that is why they still have not been implemented in SPARK, SPARK would benefit from having the possibility of specifying loop variance in its annotations, for proving the termination of non-trivial loops.

Package invariants would also benefit SPARK. SPARK is able to restrict the use of global variables in subprograms through the use of *global* and *derives* annotations, but SPARK is not able to assert that the program state has always specific properties because it lacks the notion of package invariant. Package invariants raise some implementation difficulties, such as temporary invalid states in a sequence of assignments and problems related to inheritance in OOP. These problems have been addressed by JML in the past so we are certain that they can be dealt with, especially given the restrictive nature of OOP in SPARK.

Regarding the BISL of SPARK, we could add a better way to write abstract functions, logical predicates, axioms and lemmas. ACSL is one of the most powerful BISLs when it comes to the expressiveness of these specifications. To maintain compatibility, these annotations could use a different notation so that the SPARK tools could ignore them (they would be dealt with by specific tools).

It would also be interesting to have the possibility of writing algebraic definitions in SPARK. This would enable to write things such as $top(push(stack, x)) = x$. Being able to write these types of specifications would allow for a higher level of reasoning and it would also allow to define properties about the sequence of execution (or protocol) of a given package or set of packages. The Common Algebraic Specification Language (CASL)[4] has a well-defined semantics and

could serve as a basis for this. It was developed under the rationale of specifying requirements, design and architecture of conventional software.

Another limitation of SPARK is in dealing with the verification of floating-point arithmetic (the documentation of SPARK[31] states that the realrtc option may detect numerical errors in programs but not their absence, much like testing). This is not exclusive to SPARK since the verification of floating-point arithmetic is a difficult issue (and a topic of active research) that many approaches do not even try to deal with and assume real arithmetic instead of floating-point arithmetic. There is a recent article[10] with an interesting approach to this problem. It suggests a dedicated tool to deal with the verification of the non floating-point part of the program and to use Gappa (*Génération Automatique de Preuves de Propriétés Arithmétiques*) to deal with the verification of floating-point arithmetic.

5.3 Adding Features and Tools for Certification

The certification process for critical systems relies heavily on testing, even in the presence of formal methods; this is true even for DO-178C. For JML there is a tool[11] called *jmlunit* which is capable of generating test inputs by looking at invariants and pre- and post-conditions, but it forces the user to supply predefined data, as examples for the tests.

QuickCheck[13] is another approach for generating tests. Although we have to specify generators for our custom datatypes (default datatypes have predefined generators), this definition allows for greater variation. QuickCheck seems like it could be adapted to SPARK and any implementation that would be made would benefit greatly from the strongly-typed system of the language, probably reducing the number of necessary generators that would have to be created (although it should always be possible to create test generators, for our own specific purposes, if we wish so).

It should be noted that there are several other approaches to the automatic generation of test inputs and several other tools, as for example TestEra and Korat. Because of space constraints we chose to cite only these two, since they are extremely popular at the present date and also have a large community supporting them. It should also be said that any eventual test generation could also benefit from the algebraic specification mentioned earlier by having the protocol guide the test generation process.

The approaches to testing that are described on the previous paragraphs are very close to Model-Based Testing (MBT) based on source code annotations. With this type of MBT, the model is derived from the specification in the source code and then it generates abstract test data. A model checker then checks to see if the properties of the model are being respected. In some cases, it can also be possible to generate real test data and supply it to the program as unit test.

The MBT approach is used by the Spec Explorer[35] tool, using programs written in Spec#[8] as the basis for the model. In Spec Explorer, when a counter example is found, it is shown as a graph with all the actions that took place, following the order and values that lead to the error. To do this, Spec Explorer

needs to have an automata of the program, which defines the protocol for it. This approach also allows to model check and test reactive systems. This feature may be helpful for model-checking and to do MBT on systems that may interact with sensors and/or actuators, which are an essential part of critical systems.

6 Conclusion

We have presented a short introduction to what is SPARK, the philosophy and toolset that support the language, and presented also projects, both industrial and academic, which use SPARK in their development. Furthermore, we have provided some directions to enhance SPARK that could interest people in formal verification and safety-critical software development.

We believe that the directions we suggest can bring further support to SPARK and its approach and they can be particularly useful in fostering a productive environment between the academia and industry, especially in further developing industrially usable and scalable formal methods for the (near) future.

References

1. Altran Praxis: New lunar lander project relies on SPARK programming language, http://www.altran.com/document/?f=Altran_20100610_CP_EN.pdf
2. Amey, P., Dion, B.: Combining model-driven design with diverse formal verification (January 2006)
3. Antoine, L.B., Requet, A.: JACK: Java Applet Correctness Kit. In: In Proceedings, 4th Gemplus Developer Conference (2002)
4. Astesiano, E., Bidoit, M., Kirchner, H., Krieg-Brückner, B., Mosses, P.D., Sannella, D., Tarlecki, A.: CASL: the common algebraic specification language. Theor. Comput. Sci. 286(2), 153–196 (2002)
5. Barnes, J., Chapman, R., Johnson, R., Widmaier, J., Cooper, D., Everett, B.: Engineering the Tokeneer enclave protection software. In: IEEE International Symposium on Secure Software Engineering (ISSSE). IEEE Press (2006)
6. Barnes, J.: High Integrity Software: The SPARK Approach to Safety and Security. Addison Wesley, first edn. (March 2003)
7. Barnes, J.: Safe and Secure Software: An invitation to Ada 2005. AdaCore (2008)
8. Barnett, M., Rustan, Schulte, W.: The Spec# programming system: An overview (2005)
9. Baudin, P., Filliâtre, J.C., Marché, C., Monate, B., Moy, Y., Prevosto, V.: ACSL: ANSI/ISO C Specification Language, version 1.4 (2009)
10. Boldo, S., Filliâtre, J.C., Melquiond, G.: Combining Coq and Gappa for certifying floating-point programs (2009)
11. Burdy, L., Cheon, Y., Cok, D., Ernst, M.D., Kiniry, J., Leavens, G.T., Rustan, K., Leino, M., Poll, E.: An overview of JML tools and applications (2003)
12. Chapman, R.: Industrial experience with spark. Ada Lett. XX(4), 64–68 (2000)
13. Claessen, K., Hughes, J.: QuickCheck: a lightweight tool for random testing of Haskell programs (2000)
14. Crocker, D.: Perfect developer: A tool for object-oriented formal specification and refinement (2003)

15. Guttag, J.V., Horning, J.J., Garl, W.J., Jones, K.D., Modet, A., Wing, J.M.: Larch: Languages and tools for formal specification. In: Texts and Monographs in Computer Science. Springer-Verlag (1993)
16. Homeier, P.V., Martin, D.F.: Trustworthy tools for trustworthy programs: A verified verification condition generator. In: TPHOLs. pp. 269–284 (1994)
17. Jackson, P.B., Ellis, B.J., Sharp, K.: Using SMT solvers to verify high-integrity programs. In: AFM '07: Proceedings of the second workshop on Automated formal methods. pp. 60–68. ACM, New York, NY, USA (2007)
18. Jackson, P.B., Passmore, G.O.: Proving spark verification conditions with smt solvers (December 2009)
19. Jazequel, J.M., Meyer, B.: Design by Contract: the lessons of Ariane. Computer 30(1), 129–130 (1997)
20. Jones, C.B.: Systematic software development using VDM (2nd ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1990)
21. Lau, K.K., Wang, Z.: Verified component-based software in SPARK: Experimental results for a missile guidance system. In: Proc. 2007 ACM SIGAda Annual International Conference. pp. 51–57. ACM (2007)
22. Leavens, G.T., Baker, A.L., Ruby, C.: JML: A notation for detailed design (1999)
23. Leavens, G.T., Cheon, Y., Clifton, C., Ruby, C., Cok, D.R.: How the design of JML accommodates both runtime assertion checking and formal verification. Sci. Comput. Program. 55(1-3), 185–208 (2005)
24. Marche, C., Mohring, P.C., Urbain, X.: The Krakatoa tool for certification of Java/JavaCard programs annotated in JML (2004)
25. Messer, R.: Introducing SPARK Pro 9.0 webinar (April 2010), http://www.adacore.com/home/products/gnatpro/webinars/
26. Meyer, B.: Applying "Design by Contract". Computer 25(10), 40–51 (October 1992)
27. Meyer, B.: Object-Oriented Software Construction. Prentice Hall PTR, 2nd edn. (March 2000)
28. O'Neil, I.: The formal semantics of SPARK83 (1994)
29. Owre, S., Rushby, J.M., , Shankar, N.: PVS: A prototype verification system. In: Kapur, D. (ed.) 11th International Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 607, pp. 748–752. Springer-Verlag, Saratoga, NY (jun 1992), http://www.csl.sri.com/papers/cade92-pvs/
30. SPARK Team: SPARK Examiner: The SPARK Ravenscar Profile (January 2008)
31. SPARK Team: Supplementary Release Note - The RealRTC Option (February 2009)
32. Spivey, J.M.: The Z notation: a reference manual. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989)
33. Sun microsystems: Java Card Techonolgy (2000)
34. Taft, S.T., Duff, R.A., Brukardt, R.L., Ploedereder, E., Leroy, P.: Ada 2005 Reference Manual. Language and Standard Libraries: International Standard ISO/IEC 8652/1995(E) with Technical Corrigendum 1 and Amendment 1 (Lecture Notes in Computer Science). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
35. Veanes, M., Campbell, C., Grieskamp, W., Schulte, W., Tillmann, N., Nachmanson, L.: Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer (2008)
36. Woodcock, J.: Tokeneer experiments, http://research.microsoft.com/en-us/events/vsworkshop2009/jim_woodcock.pdf
37. Yin, X., Knight, J.C., Nguyen, E.A., Weimer, W.: Formal verification by reverse synthesis. In: SAFECOMP. pp. 305–319 (2008)

Timing Analysis - From Predictions to Certificates

Nuno Gaspar[1], Simão Melo de Sousa[1,2], and Rogério Reis[2,3]

[1] RELEASE - Reliable And Secure Computation Group,
University of Beira Interior, Portugal,
{nmpgaspar, desousa}@di.ubi.pt,
[2] LIACC-UP
[3] DCC-FC, University of Porto, Portugal
rvr@ncc.up.pt

Abstract. In real-time systems, timing constraints must be satisfied in order to guarantee that deadlines will be met. The calculation of each task's worst-case execution time (WCET) is a prerequisite for the schedulability analysis, and hence of paramount importance for real-time systems. However, an accurate prediction can be difficult if the underlying hardware architecture possesses features like caches and pipelines.
In this paper we report our work in progress project on *ACCEPT*, an *Abstraction-Carrying CodE Platform for Timing validation*. Our approach counts on information gathered at source-code level (e.g. loop bounds, infeasible paths), defined by annotations that also express the intended timing behaviour. Furthermore, in the context of mobile code safety and in order to minimize the *trusted computing base*, we produce a checkable certificate whose validity entails compliance with the calculated WCET.

Keywords: real-time systems, worst-case execution time, abstract interpretation, abstraction-carrying code, fixpoint computation, timing validation

1 Introduction

Real-time systems can be seen as sets of tasks that are expected to perform some functionality under predefined timing constraints. In general, in order to ensure a correct system behaviour (schedulability analysis), an upper bound estimative for the worst-case execution time (WCET) of each task is necessary.

To improve its performance, modern processors include mechanisms like caches and pipelines which make instruction's execution time context dependent, increasing the difficulty of static timing analysis. To cope with this, one could be tempted to always assume the local worst case scenario (e.g. cache miss) in order to obtain safe predictions, but two problems could arise with such approach. On one hand, it could lead to an excessive over-approximation of the actual WCET, thus resulting in a waste of hardware resources. By other hand,

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 491–501

due to *timing anomalies* [RWT+06], the obtained prediction could in fact, be unsafe (an under-approximation).

While our platform considers the extraction of information from the source-code level (e.g. loop bounds, infeasible paths), in this paper we focus on the underlying mechanisms for the production of certificates and their validation process.

1.1 Motivation

The determination of safe and tight upper bounds for the WCET has been the object of intensive study in the literature [WEE+08]. Yet, there is no attempt, up to the present and to the best of our knowledge, to provide an independent validation mechanism w.r.t. the correctness of the predicted time bounds.

Mobile code safety has been progressively gaining notoriety in the sphere of real-time systems [SP01], both at research and industry levels, since they represent an enabling technology to tackle the limitations of standard client-server based approaches. In this context, in spite of the fact that previous methodologies are leveraged by the undertake of a formal approach, one would still have to put his faith on a potential *untrusted* third party, without being able to independently validate the correctness of the predicted time bounds.

One could argue that typically, applications loaded in embedded systems need not to satisfy real-time requirements (e.g. ring tones for mobile phones). However, in [KSH05], Kirsch et al identify the mobility of real-time programs as a challenging, but desirable feature for embedded systems. Indeed, even a software/system update can be seen as mobile code. Thus, being able to independently validate its timing behaviour would be of major interest for such systems. Moreover, a timing validation mechanism could also be a valuable asset for original equipment manufacturers and sub-contractors applications.

For instance, consider the following application scenario. There are several embedded systems that due to the functionality that they are expected to carry out, cannot be easily reachable. Coral sensors or control computers for satellites are examples of such embedded systems, where even a routine software/system update can be considered as mobile code. This updated software is also expected to satisfy stringent timing constraints, i.e. behave as mobile real-time code. Thus, being able to independently validate its timing behaviour would be of major interest for such systems.

This lack of an independent validation process is the issue that we address in this paper.

1.2 Related Work

There are numerous approaches in the literature focused in algorithms and tools for the derivation of the WCET [WEE+08]. Our goal here is not to present yet another approach of that type, but rather to emphasize how to produce a certificate that can be then used to validate the predicted time bounds.

Nevertheless, we should refer that the problem of certifying resource consumption, namely execution time, has already been addressed before, like in the work of Crary and Weirich [CW00], that use an extended type system capable of specifying and certifying bounds on resource consumption. However, this work makes no effort to determine bounds on execution times, but rather provides a mechanism to certify those bounds (for instance, obtained via a previous program analysis). The result of their approach is an executable that is certified w.r.t. resource consumption.

Furthermore, Bonenfant et al [BFHH07] present an interesting combination of information retrieved at source code level, with low-level timing information gathered with AbsInt's aiT tool [FH04]. This work provides guaranteed bounds on worst-case execution times for a strict, higher-order programming language.

The *Mobility, Ubiquity and Security* (MOBIUS) [BBC+06], and the *Mobile Resource Guarantees* (MRG) [Gua05] research projects also aim at the certification of resource consumption, their approaches rely mostly on theorem proving, whereas ours relies on abstract interpretation.

1.3 Contributions

The work reported here is included in a broader effort to provide a source level feedback on a BCET and WCET computation platform with certificate generation in the context of mobile code. This platform considers a high-level annotation language, preserving its semantics through a *WCET-aware* compilation process, and a *back-annotation* mechanism [HK07]. However, details on these features will be reported elsewhere.

In this paper, we focus on the low level interface. In this sense, this paper is a work in progress report on the BCET and WCET certificate generation and validation, and it intends to introduce and justify the underlying architecture.

1.4 Organization of the Paper

The remainder of this paper is organised as follows. Section 2 presents our architecture proposal towards an *Abstraction-Carrying Code* [APH04] platform. We explain the emergence of the certificate and how it can be used for independent validation of the calculated WCET in Section 3. Finally, conclusions and future work are discussed in Section 4.

2 Proposed Architecture

The general framework of our proposal is as illustrated by figure 1. We begin by extending the C programming language with annotations, following a *Design-by-Contract* [LC04] approach, that define the intended timing properties for each function. The timing specification of the main function is of most importance, however one may also want to define some constraints on the auxiliary functions. Moreover, by placing these annotations directly into the source code, we are also

able to express valuable informations for the subsequent WCET analysis, such as infeasible paths and loop bounds. This is achieved by the use of a WCET-aware compilation process, targeting the ARM instruction set, that preserves the annotations semantics.

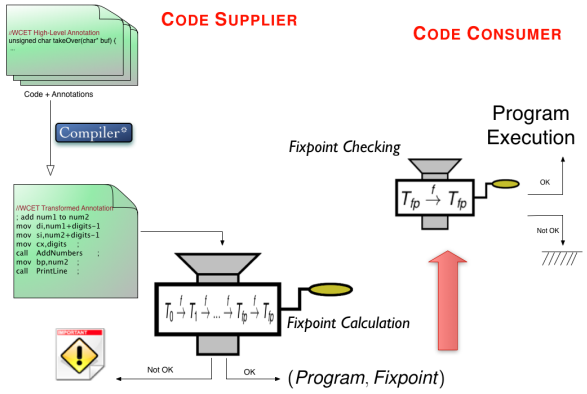


Fig. 1. Abstraction-Carrying Code based BCET/WCET Platform

Only at the hardware level one can accurately calculate the WCET, but feedback about the compliance of the given timing specification should be done at source-code level. Hence, in order to perform timing validation w.r.t. the functions' timing specification, we perform the WCET analysis at machine-code level, taking into account the effects of the hardware specific features, and alert of any possible non-compliance throughout the use of *back annotations* [HK07]. This is what is represented by the bottom left area of Figure 1. The idea of this mechanism is to propagate the timing information back to the source-code level, warning the system developers about the violation w.r.t. the timing specification. However, as stated in subsection 1.3, details on this mechanism will be reported elsewhere.

Let the set of execution times of a program $\mathcal{P}$ be denoted by $[\![\mathcal{P}]\!]$. The problem of verifying the compliance of the given timing specification can be thus formulated as follows:

$$\mathcal{P} \textit{ respects the timing specification } \mathcal{I} \textit{ if } [\![\mathcal{P}]\!] \subseteq \mathcal{I},$$

where $\mathcal{I}$ stands for the intended timing behaviour, i.e., the set of accepted execution times. The idea is to express the *collecting semantics* [WW08] of $\mathcal{P}$ as the fixpoint of a set of recursive equations. In general, however, the state space to be considered is too large to exhaustively explore all possible executions and some abstraction of the application domain is required in order to make the timing analysis feasible. With this in mind, our approach relies on abstract interpretation as the underlying technique. With its use, not only it provides us

an adequate framework to reason about, but also with an elegant way to infer an abstract model of the program that can play the role of a certificate.

2.1 Abstract Interpretation

In the abstract interpretation framework, a program $\mathcal{P}$ is interpreted over a simpler *abstract domain* $\mathcal{D}_\alpha$. This abstract domain permits to trade efficiency over precision, i.e., although it is an approximation, by computing the fixpoint over this abstract domain, we will be able to produce precise, yet safe, (over-)approximations of the collecting semantics.

The fixpoint calculation over this abstract domain will allow us to safely predict the processor behaviour for the program's execution (e.g. cache miss). With that information, and following the approach from the standard WCET architecture [Wil04], we can calculate the WCET, by determining its path through its control-flow graph. This is achieved by solving its corresponding *integer linear program* maximized for execution time. The process of determining the *best-case execution time* (BCET) is performed analogously.

Let $[\![\mathcal{P}]\!]_\alpha$ be the set of execution times calculated over the abstract domain $\mathcal{D}_\alpha$. It is clear that $[\![\mathcal{P}]\!]_\alpha$ is lower- and upper-bounded by BCET and WCET, respectively. Moreover, since the comparison between actual and intended semantics is easier if done in the same domain, we assume that the intended timing specification is also given in the abstract domain, i.e., $\mathcal{I}_\alpha \in \mathcal{D}_\alpha$.

The problem of verifying the compliance with the given timing specification can now be reformulated as

$$\mathcal{P} \textit{ respects the timing specification } \mathcal{I}_\alpha \textit{ if } [\![\mathcal{P}]\!]_\alpha \subseteq \mathcal{I}_\alpha.$$

At this stage, feedback regarding any possible non-compliance of $\mathcal{P}$ w.r.t. the timing specification can be reported through the use of back annotations [HK07], allowing the system developers to proceed accordingly.

2.2 Abstraction-Carrying Code

Proof-Carrying Code (PCC) [Nec97] is a general mechanism enabling a program consumer to locally check the validity of the code w.r.t. some safety policy. The inherent key benefit is that there is no need to trust any third party. However, there are three essential challenges for PCC to be used in practice:

(i) definition of *expressive safety and functionality policies*,
(ii) automatic generation of the certificate, i.e., proving the program correct, and
(iii) efficient certificate checking in the consumer side.

In the context of mobile code safety, most approaches rely on theorem proving, whereas Abstraction-Carrying Code (ACC) [APH04] relies on abstract interpretation.

In ACC, and in particular for the purposes of our platform, the above challenges are addressed by (i) getting hold of the effects that the processor specific

features (e.g. caches, pipeline) have on the execution time, which has already been addressed in the literature [The04,GR09]; (ii) using a fixpoint static analyzer to automatically infer an abstract model of the program, which can be then used as a certificate; and (iii) by a simple, easy-to-trust fixpoint checker.

3 Certificate Production and Validation

Let us now elaborate on this process applied to our platform proposal. A program is characterized by its control-flow graph, constituted by a set of edges $E \subseteq V \times Ins \times V$, where V represents the program points, $v_i \in V$ models the program's entry point and Ins models the instruction to be executed whenever taking that edge.

A semantic function $[\![.]\!] : Ins \rightarrow (S \rightarrow S)$ assigns to each $ins \in Ins$, a transfer function that models its effect on the program state S, being evaluated. For instance, in the ARM instruction set, the instruction B *address* is specified as $R15 := address$, i.e., update of the program counter register to the address given by the evaluation of the expression *address*, and thus would have to be modelled accordingly to its specification.

The collecting semantics assigns for each program point V, the set of program states S, which may occur in any possible execution, i.e., $CS : V \rightarrow \mathcal{P}(S)$ (where $\mathcal{P}(S)$ stands for powerset of S). The analysis to be performed can be specified by extracting a number of equations from the program being considered. There are two types of equations. The first one, relates exit with entry information for each program point V. While the second, relates entry information of a program point V_i, with exit information of nodes from which there exists an edge to the program point V_i, i.e., $\bigcup\{V_j \mid (V_j, ins, V_i) \in E\}$.

The resulting system of equations can be solved by computing the least fixpoint $lfp(F) = F^n(\lambda v.\emptyset)$ of the functional $F : (V \rightarrow \mathcal{P}(S)) \rightarrow (V \rightarrow \mathcal{P}(S))$:

$$F(f)(v') = \begin{cases} S_0 & \text{if } v' = v_{in}, \\ \bigcup_{(v,ins,v') \in E} [\![ins]\!](f(v)) & \text{otherwise,} \end{cases} \tag{1}$$

where $S_0 \subseteq S$ is the set of the program's initial states.

However, as mentioned in Section 2, computing the collecting semantics of a large and complex program $\mathcal{P}$ can be too much expensive to be feasible. Hence, the analysis is performed on a simpler abstract domain $\mathcal{D}_\alpha = (S, L, \beta, \gamma)$, where $L = (L, \sqsubseteq, \bigsqcup, \bot, \top)$ is a complete semi-lattice and $\beta : S \rightarrow L$ is a *representation function*, mapping concrete to abstract states. The idea, is that β maps a state S to the best property describing it. Finally, $\gamma : L \rightarrow \mathcal{P}(S)$ is a *concretization function* mapping abstract states to concrete states.

As we have seen above, the collecting semantics operates over sets of states, while our abstract domain, operates over sets of properties. Thus, with the purpose of relating these two domains, we define an *abstraction function* $\alpha : \mathcal{P}(S) \rightarrow L$, by $\alpha(S') = \bigsqcup\{\beta(s) \mid s \in S'\}$. The concretization function γ, and the abstraction function α, will therefore yield the following relation:

$$\mathcal{P}(S) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} L$$

The above relation is defined such that $\alpha(X) \sqsubseteq l \Leftrightarrow X \subseteq \gamma(l)$, and thus establishing the pair (α, γ) as a *Galois connection*. Furthermore, in order to ensure termination we require the *Ascending Chain Condition* to hold, i.e., every ascending chain of elements eventually terminates. For this, both the abstraction function α, and the concretization function γ, must be monotonic w.r.t. the $\sqsubseteq$ and $\subseteq$ operators, respectively.

The semantic function defined above, can now be redefined as an *abstract semantic* function $[\![.]\!]_\alpha : Ins \to (L \to L)$, over the abstract domain. The abstract counterparts of the transfer functions $[\![ins]\!]$, i.e., $[\![ins]\!]_\alpha$, must also be monotonic w.r.t. the $\sqsubseteq$ operator. Finally, the analysis can now be applied with the *abstract collecting semantics* $CS_\alpha : V \to L$, such that $\forall v \in V : CS(s) \subseteq \gamma(CS_\alpha(v))$, i.e., the computed results are either precise or an over-approximation of the collecting semantics, and thus are safe.

The resulting system of equations can be solved by computing the fixpoint $lfp(F_\alpha) = F_\alpha^n(\lambda v.\bot)$ of the functional $F_\alpha : (V \to L) \to (V \to L)$:

$$F_\alpha(f)(v') = \begin{cases} l_0 & \text{if } v' = v_{in}, \\ \bigsqcup_{(v,ins,v') \,\in\, E} [\![ins]\!]_\alpha(f(v)) & \text{otherwise,} \end{cases} \tag{2}$$

where the abstraction of the *concrete* initial states is defined as initial abstract state, thus $\alpha(S_0) \sqsubseteq l_0$.

It should be clear that, since the abstract transfer functions, $[\![ins]\!]_\alpha$, are monotonic w.r.t. the $\sqsubseteq$ operator, by induction we obtain $F_\alpha^n(\lambda v.\bot) \sqsubseteq F_\alpha^{n+1}(\lambda v.\bot)$ for all n. All the elements of the sequence are in L, and since this is a finite set, not all elements of the sequence can be distinct. Thus, there must be some n such that:

$$F_\alpha^{n+1}(\lambda v.\bot) = F_\alpha^n(\lambda v.\bot)$$

Furthermore, since $F_\alpha^{n+1}(\lambda v.\bot) = F_\alpha(F_\alpha^n(\lambda v.\bot))$, we have reached the least fixpoint of F_α, i.e., $lfp(F_\alpha)$, and thus found a solution to the equation system.

The analysis to be instantiated depends on the target processor being evaluated. In our current prototype implementation of *ACCEPT*, we focus ourselves in the ARM7TDMI-S and ARM920T processors. While for the former only a pipeline analysis is performed that captures the instruction's overlapping effect, for the latter, since it also features a cache memory, an integrated cache and pipeline analysis is performed [The04].

3.1 Program Producer - Certificate Production

After computing this fixpoint, and thus having the cycles counts for each basic block of the control-flow graph, we are able to calculate both the BCET and

WCET by means of integer linear programming techniques, and thus verify the compliance with the giving timing specification (Figure 1). However, we can also let the obtained fixpoint play the role of a certificate.

In the context of mobile code safety one cannot trust the origin of the program. Hence, by adding to the code the certificate and sending both to the program consumer, it can be performed a local and independent check of the program's timing behaviour, thereby avoiding the need to trust in the code producer.

3.2 Program Consumer - Certificate Validation

The program consumer receives a program along with its certificate. In order to check the compliance with the intended timing specification, the first step is to compute the program's control-flow graph and verify that the certificate is a valid *abstraction*. Then, since the certificate is supposedly a fixpoint, the checking procedure can be written as:

$$Check(Certificate) = \begin{cases} True & \text{if } F_\alpha(Certificate) = Certificate, \\ False & \text{otherwise.} \end{cases} \quad (3)$$

Since the certificate is supposed to be a fixpoint, another iteration over it cannot change anything, thus, on the program consumer side, a simple one-pass computation is suficient to check that the certificate is indeed a fixpoint.

In the cases where the received certificate does not behave as a fixpoint, the program consumer can simply reject the program. One could argue that we could let the program run, and *kill* its execution in the case of a timing behaviour non-compliance. However, that would be a waste of resources, and in the context of embedded systems, which tend to have very limited computational resources, it is unacceptable. On the other hand, if the certificate is indeed a fixpoint, then the program consumer can locally compute the BCET and WCET by standard integer linear programming techniques, and thus check the compliance with the timing specification. Furthermore, it should be noted that in this framework, it is also possible for the program consumer to define new timing policies. For instance, one can be interested in tightening the timing constraints.

This validation process requires that both the producer and consumer share the same abstract transfer functions. Indeed, if the consumer used different abstract transfer functions the certificate checking process would be inefficient, and thus prohibitive for such scarce resource equipments as embedded systems. One could argue that the independence in the timing validation process is compromised by that fact, however, it should be noted that the trusted computing base is limited to this checking operation, i.e., a simple, easy-to-trust fixpoint checker, that only has to perform a one iteration process. Hence, this approach allows to detect if a program has been tampered with, since an adulteration in the program code would be detected when performing the checking operation, i.e.,

the fixpoint iteration. In the context of mobile code, this is particularly relevant since, rather than simply put a blind confidence on a previous timing analysis, one can validate the program's timing behaviour by solely relying on a fixpoint checker.

4 Conclusions and Future Work

Abstract Interpretation has been widely used in the industry, being static timing analysis one of its most successful applications [WW08]. In our approach we also use the Abstract Interpretation framework as the underlying technique, we obtain our BCET and WCET predictions taking into account the hardware specificities (cache, pipeline) [The04,GR09], by explicitly following a standard fixpoint computation strategy [Kil73], and then apply standard integer linear programming techniques in order to compute the BCET and WCET. This fixpoint computation will allow us to infer an abstract model of the program, which can then be used as a certificate, i.e., a program consumer can locally validate the received program w.r.t. to its timing behaviour, by simply checking that this abstract model is indeed a fixpoint (a one-pass process), and then compute the BCET and WCET with the received certificate.

This paper is a work in progress report on the timing certificate generation and validation and intend to introduce and justify the underlying architecture. We presented our architecture proposal for *ACCEPT*, an *Abstraction-Carrying CodE Platform for Timing validation*. In our prototype being implemented, we avoid a *binary-to-assembly* translation phase, by making our compilation process directly produce ARM assembly.

At this stage there are still some open issues that remain to be addressed. One of the main challenges that we face in order to make our *ACCEPT* platform useful in practice is the size of the produced certificates. Embedded systems are known for their scarce resources, and thus, cannot afford to waste computational means. In [AASPH06], Albert et al introduce the notion of a *reduced certificate*, with the objective of producing a certificate that only contains the essential information which the program consumer cannot reproduce by itself, while not yielding an overhead in the certificate checking process.

Our actual focus on the certification generation part of the *ACCEPT* platform is now on the pragmatical evaluation of our proposal. For now we are not concerned with performance, but with correctness and adequacy (in the context of mobile code). However, a BCET/WCET platform is only useful if it provides tight and safe time bounds. In this sense, we use state of the art algorithms for their calculation. Nevertheless, comparing equitably this kind of platform against reference tools [FH04], even pragmatically in the form of benchmark, is not a trivial task, the same source-code, compilation process and/or low-level code and target architecture must be considered. However, we plan to report on case studies and practical results of our framework very soon.

To the best of our knowledge this is the first work applying the concepts of Abstraction-Carrying Code to the static timing analysis field.

Acknowledgments

This work is partially supported by the RESCUE project PTDC/EIA/65862/2006 funded by FCT (Fundação para a Ciência e a Tecnologia). We also thank Vitor Rodrigues for his valuable suggestions and comments.

References

[AASPH06] Elvira Albert, Puri Arenas-Sánchez, Germán Puebla, and Manuel V. Hermenegildo. Reduced certificates for abstraction-carrying code. In *ICLP*, pages 163–178, 2006.

[APH04] Elvira Albert, Germán Puebla, and Manuel V. Hermenegildo. Abstraction-carrying code. In *LPAR*, pages 380–397, 2004.

[BBC+06] Gilles Barthe, Lennart Beringer, Pierre Crégut, Benjamin Grégoire, Martin Hofmann, Peter Müller, Erik Poll, Germán Puebla, Ian Stark, and Eric Vétillard. Mobius: Mobility, ubiquity, security. In *TGC*, 2006.

[BFHH07] Armelle Bonenfant, Christian Ferdin, Kevin Hammond, and Reinhold Heckmann. Worst-case execution times for a purely functional language. In *In 18th IFL 2006*. Springer, 2007.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77*, pages 238–252. ACM, 1977.

[CW00] Karl Crary and Stephnie Weirich. Resource bound certification. In *POPL '00*, pages 184–198, New York, NY, USA, 2000. ACM.

[FH04] Christian Ferdinand and Reinhold Heckmann. ait: worst case execution time prediction by static program analysis. In *IFIP Congress Topical Sessions*, pages 377–384, 2004.

[GR09] Daniel Grund and Jan Reineke. Abstract interpretation of FIFO replacement. In *SAS'09*, pages 120–136. Springer-Verlag, 2009.

[Gua05] Mobile Resource Guarantees. http://groups.inf.ed.ac.uk/mrg/, 2005.

[HK07] Trevor Harmon and Raymond Klefstad. Interactive back-annotation of worst-case execution time analysis for java microprocessors. In *13th IEEE RTCSA*, Washington, 2007. IEEE Computer Society.

[Kil73] Gary A. Kildall. A unified approach to global program optimization. In *POPL '73*, pages 194–206, New York, NY, USA, 1973. ACM.

[KSH05] Christoph M. Kirsch, Marco A. A. Sanvido, and Thomas A. Henzinger. A programmable microkernel for real-time systems. In *VEE '05*, 2005.

[LC04] Gary T. Leavens and Yoonsik Cheon. Design by contract with jml, 2004.

[Nec97] George C. Necula. Proof-carrying code. In *POPL '97*, pages 106–119, New York, NY, USA, 1997. ACM.

[RWT+06] Jan Reineke, Bjrn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. A definition and classification of timing anomalies. In *6th Intl Workshop on WCET Analysis*, 2006.

[SP01] Alexander D. Stoyen and Plamen V. Petrov. Towards a mobile code management environment for complex,real-time, distributed systems. *Real-Time Syst.*, 21(1/2):165–189, 2001.

[The04] Stephan Thesing. *Safe and Precise WCET Determination by Abstraction Interpretation of Pipeline Models.* PhD thesis, Saarland University, 2004.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–53, 2008.

[Wil04] Reinhard Wilhelm. Why ai + ilp is good for wcet, but mc is not, nor ilp alone. In *VMCAI'04, LNCS 2937*, 2004.

[WW08] Reinhard Wilhelm and Björn Wachter. Abstract interpretation with applications to timing validation. In *CAV '08*, 2008.

Towards a Formally Verified Kernel Module

Joaquim Tojal[1,2], Carlos Carloto[1], José Miguel Faria[2], and Simão Melo de Sousa[1,3⋆]

[1] University of Beira Interior,Dept. of Computer Science, Covilhã, Portugal
[2] Critical Software SA.,Coimbra, Portugal
[3] University of Porto, Artificial Intelligence and Computer Science Laboratory (LIACC-UP), Porto, Portugal

Abstract. In this article we present the design by contract approach to formal verification of an industrial real-time kernel using VCC (Verified C Compiler) and Frama-C tools. The annotations were directly inserted into the source code of an industrial kernel module, xLuna, and verified automatically. VCC was also used to reason about concurrency issues in a preemptable and real-time environment. In addition we describe some particular methodological aspects of these two verifiers. These are the first results towards a Formally Verified Kernel.

1 Introduction

Almost every computer system depends directly on the operating systems behavior. As such, having kernel code that is proved to be correct is a goal that researchers and industrial companies have attempted to achieve. Large amounts of low-level implementations like operating system core is obviously a perfect and challenging target for formal verification. The interest in formally verifying realistic and industrial low-level code and obtaining the highest standards of safety like Common Criteria EAL7 has grown significantly in recent years. In this work we target precisely this goal taking a modular approach to formally verify a real industrial real-time operating system kernel which was not designed with formal verification in mind. The kernel targeted is a particular interrupt manager of xLuna real-time kernel for embedded systems built by Critical Software,SA. For verification, we use Microsoft Research Verified C Compiler(VCC) and Frama-C tools to reason about functional correctness, concurrency and safety properties of xLuna kernel. The specifications are based in Hoare-style pre- and post-conditions inlined with the real code.

The remaining of this paper is organized as follows: Section 2 show the xLuna architecture and some particular design aspects. Then, in section 3, we gave an overview toolchain and verification methodologies for both, VCC and Frama-C. Section 4 exposes the detail design of xLuna IRQ manager and respective verification approaches. At the end, sections 5 and 6 give some conclusions that led us to future improvements and related work to the subject.

⋆ This work was partially funded by Fundação para a Ciência e Tecnologia (FCT) and Program POSI and the project RESCUE (PTDC/EIA/65862/2006)

2 xLuna

xLuna [17] is a microkernel based on the RTEMS [18] Real-Time Operating System with the ability to run a GNU/Linux Operating System [19], providing therefore a runtime environment for real-time (RTEMS) and non-real-time (Linux) applications. xLuna was designed to supports ESA's LEON SPARC processor [17] with the main goal of extending the RTEMS kernel in order to enable a safely Linux execution without jeopardizing aspects of reliability, availability, maintainability and safety. Its general architecture is shown in Figure 1. The

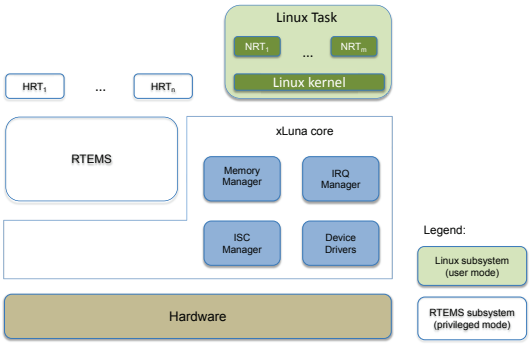


Fig. 1. xLuna architecture

Linux subsystem runs as an unprivileged RTEMS task and in a different memory partition. This provides spacial partitioning, guaranteeing a safe isolation between the non-real-time (NRT) and hard-real-time (HRT) subsystems. xLuna provides four main modules:

- *Memory manager*: enforces the isolation requirements between RTEMS and Linux and memory protection of Linux kernel from NRT user processes.
- *Interrupt Manager (IRQ)*: connects interrupts of the real hardware to the Linux kernel (which does not have access to them) and provides services.
- *Inter-System communication (ISC)*: for bidirectional communication between HRT and NRT tasks.
- *Device Drivers*: for other hardware virtualizations required (e.g. timer).

In the following section we describe the important aspects of the verification tools used on xLuna's kernel.

3 Methodology and Tools

3.1 Verified C Compiler (VCC)

VCC [11] is an automatic verification tool for concurrent C that is being developed by Microsoft Research, Redmond, USA and European Microsoft In-

novation Center (EMIC), Aachen, Germany. VCC utilizes annotations based in Hoare-style pre- and post-conditions closely attached to source code. That is, the annotations are mixed into the *codebase* rather than within blocks of commented (for the C compiler) specifications. Even so, the annotated C code can still be compiled with any C compiler through conditional compilation: If a regular C compiler is called, a special VCC flag is disabled and the annotations and specification code are preprocessed and transformed into empty strings. The concept of complete code with annotations is the same used in JML [16] for Java code or SPEC# [13] for C# programs.

Verification Toolchain VCC is fully integrated with the well known Microsoft Visual Studio IDE, providing a familiar environment to programmers and making the correct use of the VCC toolchain very simple. Three main steps are made by VCC when trying to verify an annotated C program: (i) VCC translates annotated C code into BoogiePL (an intermediated verification language), (ii) Boogie translates BoogiePL into first-order predicate formulas (verification conditions) and (iii) Z3 tries to solve them and if it finds a proof then the program is correct according to the specifications.

VCC Annotations As in standard *design by contract* approach to partial correctness, the pre- and post-conditions inserted into a C function constitute a contract between the function and a function caller, guaranteeing that if the function starts in a state that satisfies the precondition then the postcondition holds at the end of the function. To express and reason about specifications, VCC uses clauses such *requires*, *ensures*, *result* or *writes* representing, respectively, pre- and post-conditions, function return value, and frame conditions which limit what the function is authorized to modify. VCC also supports loop invariants for reasoning about loop behavior and termination.

Ownership, Type Invariants and Ghost Code VCC memory model ensures that objects (pointers to structures) do not overlap in memory keeping a typed and hierarchical view of all objects (Spec# ownership model). This ownership machinery is applied as a *ghost* transformation behind every structure in the program. Each type has a related ownership control object in specification code (*ghost* code), only visible to VCC and providing a bridge between implicit specification code and VCC annotations in the real program. For the sake of brevity the transformation process is not detailed in this paper (a more detailed account can be found in [21], available at `http://www.di.ubi.pt/~release`). *Ghost* code can also exist as explicit specification functions, objects or variables only seen by VCC for verification purposes. Structures can be also annotated with invariants related to their ownership behavior or to their own fields.

VCC implicitly lets an object own its representation and writing the object allows writing its ownership domain. However, updating an object requires a special *wrap/unwrap* protocol to transfer ownership domains (see Figure 2).

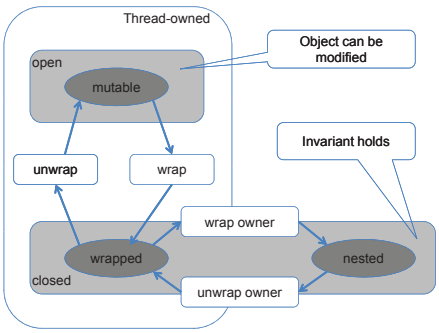


Fig. 2. VCC wrap/unwrap protocol

Defining an hierarchical structure gives VCC a tree view of the system. One object can only have one owner and threads own themselves. Figure 2 shows that in a specific ownership domain each object should be *open* or *close* inside of the thread domain and *close* outside the thread domain. In a sequential fashion an object can only be updated when it is *mutable* and must return to a safe state performing some operations required by the *wrap*/*unwrap* protocol. The explanation of these states is given bellow.

- **Mutable**. After creation the object is open and owned by *me*() which represents the current thread. In this state the thread is allowed to modify the object and prevents other threads interference.
- **Wrapped**. Wrapping the object (*wrap*) requires its invariant to hold. The reverse transition (*unwrap*) assume object invariants.
- **Nested**. Closed objects can be added to (or removed from) another objects ownership via *set_owns*(), *set_closed_owner*() (wrap owner) or unwrap owner operations using *giveup_closed_owner*(). The reverse transition (*unwrap*) assume object invariants.

3.2 Frama-C

Frama-C is a platform dedicated to the analysis of software source code written in C. It gathers a series of different tools with several static analyses techniques and a deductive verifier in a single collaborative framework. Its collaborative approach allows different analyzers to build upon the results previously computed by other analyzers. Frama-C is an extensible framework. It is open source software and is organized with a plugin architecture. It contains several ready-to-use plugins and new plugins may be built and use the results or functionalities provided by the existing plugins. A common kernel centralizes information and conducts the analysis. Plugins interact with each other through interfaces defined by the kernel.
Currently, Frama-C provides several *lightweight* analyzers (e.g., "Metrics", "Call-graphs", "Users", "Constant Folding", and "Occurrence"), semantic analyzers

(e.g., “Functional Dependencies”, “Slicing”, and “Impact”), the sophisticated “Value Analysis” plugin, which automatically computes variation domains for the variables of a program, and deductive verification, through the “Jessie” plugin. Jessie is the Frama-C plugin that enables design by contract development of C programs. The contracts are written in the ANSI/ISO C Specification Language (ACSL) [15]. The generated verification conditions can be submitted to external automatic provers such as Simplify, Alt- Ergo, Z3, Yices, and CVC3.

4 Verification of xLuna IRQ Manager

Verifying low-level code that was not implemented with formal verification in mind and adapting the verification methodology to that implementation is a non-trivial task. When reasoning about xLuna as a formal verification target its own architecture suggests that a modular approach should be taken in order to achieve an overall correctness proof. One of the most critical and crucial modules of xLuna is the IRQ manager, since it has to catch and process all interrupt requests needed for proper system functionality. It thus constitutes an interesting target for formal verification and was the subject of our study. Yet, the different modules are considerably dependent on each other. Moreover, kernel code is highly dependent of machine assembly instructions, which causes issues for the verification task since VCC does not interpret assembly language. For this reason, it was decided, at this stage, to assume that machine instructions and inlined assembly are correct. All IRQ module dependencies were studied to build proper code isolations and abstractions were made to fit the verification methodology.

4.1 IRQ Design

Following xLuna architecture, the Linux kernel is running as an unprivileged (user-mode) RTEMS task and thus, it does not have direct hardware access. The main purpose of the IRQ manager is to serve as a bridge connecting the Linux subsystem to hardware interrupts. This enables catching incoming hardware interrupts required by Linux kernel, system calls made by Linux processes, or xLuna system calls made by Linux kernel. Hardware interrupts or software traps are both called *events* that are inserted into an *Event Queue* to be sent to their handlers.
Interrupts are filtered by a dispatcher function which is responsible to insert them into the event queue. Once there are events in the queue they should be processed by the IRQ manager through a *monitor task* which calls the respective Linux handlers or xLuna services (see Figure 3). Events can be treated synchronously or asynchronously. In the queue structure there can be only one *sync* event at a time. This event is a request caused by the running instruction and must be processed synchronously. *Async* events are accumulated in the queue according to their event type (e.g., TT_ISC_RTEMS_TO_LX is the special trap type 0x21 for inter systems communication that is inserted as an async event). As shown in

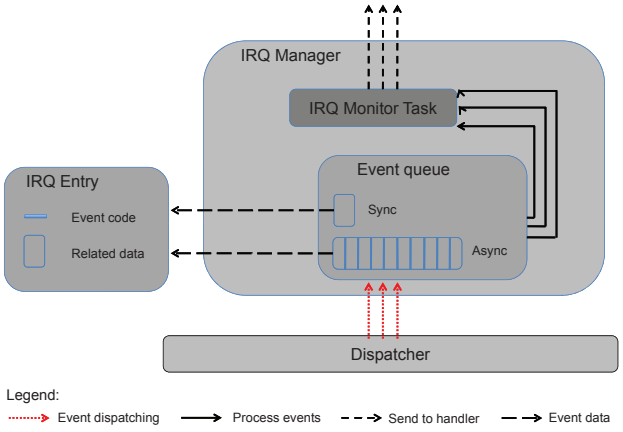


Fig. 3. Interrupt request manager

Figure 3, both *Sync* and *Async* events have an associated data structure which contains the data needed for event handlers and the interrupt/trap code.

In the next sections, we describe the approach taken for verifying the treatment of each event inserted in the queue and the correct usage of data structures evolved in this flow.

4.2 VCC Verification Approach

We can drive the verification methodology used, through the analysis of an IRQ manager function used to insert a synchronous event into the queue. As already mentioned, VCC enforces a ownership model to guarantee a consistent and typed view of all objects (e.g., pointers to data structures) which ensures for instance that objects of the same type and different addresses do not overlap in memory [12].

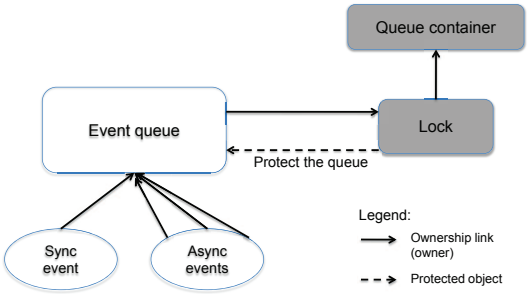


Fig. 4. VCC event queue ownership

When xLuna interrupt manager is initialized, VCC considers all objects as mutable. Therefore ownership relations must to be configured at the program entry point (*irq_init()* function) to prevent further object updates without complying with the ownership protocol. This setup ensures at IRQ manager boot time that:

- The queue is *typed* and protected at the end of the initialization;
- The queue is the owner of all *sync* and *async* events;
- When the queue is *closed* its invariant and the invariants of all embedded types hold.

With this ownership setup in mind, the following example illustrates the addition of a *sync* event to the queue. The code is a simplified version of the original function.

```
1  int event_queue_insert_sync(struct irq_entry *irq claimp(c))
2    always(c,closed(&QueueLock))
3    requires(nested(&event_queue))
4    requires(irq->data != NULL)
5    requires(!VCC_SPEC_FUN_is_asyn_trap(irq->event))
6    requires(thread_local(irq))
7    reads(irq)
8    writes(&event_queue)
9    ensures(&event_queue.sync_event == irq)
10   ensures(nested(&event_queue))
11   {
12   //xLuna assert function
13   ASSERT(irq->event);
14     assert(typed(&irq->event));
15   sparc_disable_interrupts();
16     spec(VCC_SPEC_FUN_sparc_disable_interrupts(spec(&QueueLock
         ) spec(c));)
17     // unwrap/wrap protocol
18     unwrap(&event_queue);
19     unwrap(&event_queue.sync_event);
20   //insert sync event(for vcc we can now change the queue)
21   event_queue.sync_event = *irq;
22     //reverse wrapping
23     wrap(&event_queue.sync_event);
24     wrap(&event_queue);
25   sparc_enable_interrupts();
26     spec(VCC_SPEC_FUN_sparc_enable_interrupts(spec(&QueueLock)
         spec(c));)
27   return 0;
28 }
```

Listing 1.1. Insert synchronous event function in VCC

As preconditions one tells to VCC that this function will start in a state where:

- *event_queue* is owned by an object and thus its invariant holds (line 3);

– *sync* events always have *data* required by event handlers to work properly (line 4);
– event code must be within sync event bounds (line 5);
– *irq* points to an object that is local to the running thread (lines 6) which is only allowed to read the same *irq* object (lines 7). On the other hand, permissions to change *event_queue* are necessary. The *writes* clause guarantees that only *event_queue* will be updated (line 8).

To respect the VCC ownership protocol one needs to *unwrap* (lines 18, 19) the queue and its embedded IRQ entry, change it and then *wrap* in reverse order (lines 23, 24). When an object is *unwrapped*, VCC implicitly *assume* its invariant and *assert* it when wrap occurs. In the example, this ownership flow will guarantee the second postcondition (line 10), whereas the first one ensures that the queue *sync* event was updated correctly (line 9).

Lines 2, 15, 16, 25, 26 and the ghost claim parameter (line 1) are related with concurrency verification and are explained in next section.

4.3 Concurrency Verification

The above described approach is suitable for sequential code. Nevertheless, the kernel has to guarantee that executions are made without interference or unexpected kernel behavior. In a concurrent real-time kernel, not only user processes are preemptable but also kernel processes may be subject to stringent scheduling policies or interrupts. In xLuna's implementation, in order to achieve non-interference updates of critical regions, low-level functions (implemented in assembly language) that disable and enable back interrupts are used. The required update steps are performed while interrupts are disabled. In our verification approach, such assembly functions are represented by VCC ghost code instructions.

Lock approach Disabling interrupts gives to the running thread non-interference execution steps before interrupts are enabled again. One can think about this low-level access policy as a mandatory lock and use an abstract ghost implementation suitable for VCC methodology. When interrupts are disabled we call a VCC ghost specification function (see Listing 1.1 line 16) that will transfer ownership domain to the running thread and prevent preemption while interrupts are enable (line 26). This allows reasoning about implementation steps in a sequential fashion. Still, one needs to ensures that the lock is not destroyed or deallocated: in VCC we can model this issue through a *claim*. The gray part of Figure 4 shows the ownership configuration for ghost code and states that the *Queue Container* ghost structure is the owner of the lock and the latter owns the queue. The *container* has two invariants saying that (i) the object protected by the lock is the *queue* and (ii) it is the owner of the lock. In VCC this knowledge can be passed to functions through ghost parameters (claims). In the example, the ghost claim parameter and the clause in line 2 guarantee that:

– c has an implicit invariant ensuring that the *container* remains *closed* (*container* invariant holds)
– the *always* clause tells VCC to enforce the fact that if the *container* is *closed*, the lock is also *closed* and thus it can never be destroyed.

4.4 Frama-C Verification Approach

The main focus of the Frama-C verification was the safety and functional issues of xLuna IRQ Manager. Before that, the initial stage comprised the evaluation and tailoring of the original code: since, at the moment, Frama-C does not support all the C language, some functions had to be changed to resolve some incompatibilities. The main topics covered were:

– *Pointer dereferencing*: the code of the xLuna IRQ Manager has a significant number of global variables and pointers. As such, pointer dereferencing was one of the most relevant aspects;
– *Arguments*: analyze if the function arguments are correctly introduced and do not prevent the functions from terminating;
– *Loops*: analyze the body of the loops and build the necessary contracts (loop variants and loop invariants) to prove that each loop terminates

To illustrate the Frama-C verification approach, we consider the same example, i.e., the function that inserts synchronous events into the event queue.

```
1  /*@ requires \valid(irq);
2    requires (irq->data!=NULL);
3    ensures \result==0 || \result==(-1);
4      @*/
5  int event_queue_insert_sync(struct irq_entry *irq)
6  {
7    int level;
8    ASSERT(irq->event);
9    if (event_queue_has_sync_event()) return -1;
10   //@ assert \valid(irq);
11   //@ ensures event_queue.sync_event == *irq;
12   event_queue.sync_event = *irq;
13   return 0;
14 }
```

Listing 1.2. Insert synchronous event function in Frama-C

The contracts included in this function are explained as follows:

– Line 1: This contract is a precondition: in order to the function finishes it needs a valid irq. It is required that irq pointer is allocated in a safely memory location.

– Line 2: Also a precondition. In this contract we say that the data cannot be null.
– Line 3: A postcondition that guarantees that the output of the function is either 0 or -1. (It is 0 if the event is inserted; if not the output is -1.)
– Line 8: This ASSERT is not part of the contracts that we build. It is an original xLuna C statement.
– Line 10: An assertion. It strengthens the precondition on line 1. It is right before the place where is absolutely necessary that irq is not NULL.
– Line 11: The postcondition of the function.

For this function, it were generated 15 proof obligations; all of them proved with success by the automatic prover.

The proof obligations were generated by why, and Alt-ergo was the automatic prover utilized to discharge them. In the total of the project, it were generated 373 proof obligations., all of them automatically proved by Alt-ergo. The results are analyzed in the next section.

5 Conclusions and Future Work

In this paper we have presented the design by contract approach to formal verification of a realistic system using different verification tools for a feasibility study. One can instantly conclude that the annotations burden in VCC is greater than Frama-C, mainly due to the ghost code and type invariants supported by VCC. In the overall IRQ model (about 1k LOC C) were inserted almost ten times more VCC annotations than Frama-C. Safety properties such as correct array index, arithmetic overflow and pointer deference or null pointers were verified in most parts of the IRQ manager. All inside function updates were surrounded by frame conditions and all parameters validated, type invariants hold with respect to the VCC ownership protocol. As said before VCC memory model guarantees that objects do not overlap in memory. At this time, pre- and post-conditions were added to approximately 80% of IRQ manager C code. The rest of the IRQ C code is related to switching and Linux stack manipulations. The concurrency verification approach guarantees sequential execution in a preemptable environment and it is proved to be suitable to VCC methodology [20,10] and also used in PikeOS. However, assembly language in kernel code can not be ignored when aiming to an overall system verification. One possibility to extend VCC work is the construction of a ghost model of the underlying hardware and assembly language, and connect the specifications made to the ghost model.

After this work we are able to do an analysis of the platform Frama-C. It is a platform that is being developed and in the last months it has been in constant evolution. In this moment it is a platform that is good to work with. The integration of the ACSL, Jessie and the why platform makes Frama-C a very good platform to work in the verification of programs. However it has some problems that the developer needs to improve. Some of those problems are:

– Frama-C has limited support for function pointers. This is a problem when used on the type of code found in an operating system module.

– It currently lacks clear error messages describing what and where the problem is.
– Its implementation is not stable yet (it is an academic tool), it crashes often. This mostly happens when there are pointers present in the code to be verified.
– Some annotations cause crashes of Frama-C or its subsystems.

With Frama-C, in the functions that we analyze, the results on the automatic prover was a success, and all the proof obligations of those function were checked with success. As said before, some code of a few functions were removed because of Frama-c incompatibilities, and other functions weren't analyze because of completely incompatibility with Frama-C. We have verified about 80% of the IRQ Manager code. So as the Frama-C development proceed it will be possible to reintroduce the code removed and to analyze the functions that we couldn't analyze with the actual version of Frama-C. After the work done in the verification of the xLuna IRQ Manager we should be able to proceed to other xLuna modules. As long as the xLuna modules aren't too incompatible with the actual version of Frama-C the verification of those modules may be possible. However with the evolution of Frama-C it may be possible to verificate all of xLuna modules.

6 Related Work

Prove correctness of low-level software implementations is a goal pursued for decades, the early work on formal verification of operating systems comes from 1973-1980 with the Provably Secure Operating System (PSOS) [1]. Later in seventies UCLA Data Secure Unix (DSU) [2] was the first approximating the modern microkernel architecture. The proofs were guided based on first-order predicate calculus and 20% of the code have been proven correct. Other early work is the Kernel for Isolated Tasks (KIT) [3], KIT address the problem of verifying properties for process isolation in a multi-tasking environment. PSOS, DSU and KIT were pioneers in attempts to large scale software verifications and inspired some techniques still used nowadays. Verified Fiasco (VFiasco) [4] project started in 2001 with a experiment using SPIN model checker to verify a small version of the Fiasco inter-process communication. After this experiment the project moved on using the PVS theorem prover to formalize a subset of C++ to reason about Fiasco implementations. SPIN model checker was also used in other kernels such as Fluke [5], RUBIS [6] or HARMONY [7]. The L4 micro-kernel has also been a target for formal verification projects, the most recent in seL4 project [8]. seL4 was concluded at the end of 2007 with a resulting small (8700 LOC C and 600 LOC assembly) microkernel for run in ARM architecture. The OS design team used Haskell and Isabelle/HOL for fast prototyping and specification proofs. More within the scope of this paper are the VerisoftXT project which uses VCC verification methodology to prove correctness of Microsoft Hyper-V Hypervisor [9] and SYSGO PikeOS microkernel [10].

References

1. P. Neumann, R. Feiertage: PSOS Revisited. Proceedings of the 19th Annual Computer Security Applications Conference (2003) 208
2. B. Walker, R. Kemmerer, G. Popek: Specification and verification of the ucla unix security kernel. Commun. ACM (1980) 118-131
3. W. Bevier: A verified operating system kernel. Report 11, Computational Logic Inc., Austin, Texas (1987)
4. M. Hohmuth, H. Tews: The vfiasco approach for a verified operating system. In 2nd ECOOP Workshop on Program Languages and Operating Systems (2005)
5. P. Tullmann, J. Turner, J. McCorquodale, J. Lepreau, A. Chitturi, G. Back: Formal methods: A practical tool for os implementors. Proceedings of the 6th Workshop on Hot Topics in Operating Systems (1997)
6. G. Duval, J. Julliand: Modeling and verification of the rubis microkernel with spin. In Proceedings of the First SPIN Workshop (1995)
7. T. Cattel: Modelization and verification of a multiprocessor real-time OS kernel. Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques VII (1995)
8. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, S. Winwood: seL4: Formal verification of an OS kernel. In Proc. 22nd ACM Symposium on Operating Systems Principles (SOSP) (2009) 207-220
9. D. Leinenbach, T. Santen: Verifying the Microsoft Hyper-V Hypervisor with VCC. Refinement Based Methods for the Construction of Dependable Systems, number 09381 in Dagstuhl Seminar Proceedings (2010) 104-108.
10. C. Baumann, B. Beckert, H. Blasum, T. Bormer: Ingredients of Operating System Correctness. Embedded World 2010 Conference (2010)
11. E. Cohen , M. Dahlweid , M. Hillebrand , D. Leinenbach , M. Moskal , T. Santen , W. Schulte , S. Tobies: VCC: A Practical System for Verifying Concurrent C. Theorem Proving in Higher Order Logics (2009)
12. E. Cohen, M. Moskal, W. Schulte, S. Tobies: A Precise Yet Efficient Memory Model for C. 4th International Workshop on Systems Software Verification (2009)
13. M. Barnett, K. Leino, and W. Schulte: The Spec# programming system: An overview. In CASSIS 2004, LNCS vol. 3362, Springer (2004)
14. Frama-C Web page: `http://frama-c.com/`
15. J. Burghardt, J. Gerlach, K. Hartig, J. Soto, C. Weber: ACSL By Example: Towards a Verified C Standard Library (2010)
16. G. Leavens, and Y. Cheon: Design by Contract with JML. JML tutorial (2006)
17. P. Braga, L. Henriques, M. Zulianello: xLuna:eXtending free/open-source reaL-time execUtive for oN-bord space Applications. Small Satellites Systems and Services The ESA 4S Symposium (2008)
18. RTEMS web page: `http://www.rtems.com/`.
19. Snapgear Embedded Linux web page: `http://www.snapgear.org/`.
20. E. Hillebrand, D. Leinenbach: Formal Verification of a Reader-Writer Lock Implementation in C. 4th International Workshop on Systems Software Verification (2009) 123-141
21. J. Tojal. Towards a Formally Verified Microkernel using the VCC Verifier. Master's thesis, University of Beira Interior, Portugal (2010)

Inferência de tipos em Python ⋆

Eva Maia Nelma Moreira Rogério Reis
{emaia,nam,rvr}@ncc.up.pt

DCC-FC & LIACC -UP

Resumo As linguagens dinamicamente tipificadas, como a linguagem Python, permitem ao programador uma maior flexibilidade, no entanto privam-no das vantagens da tipificação estática, como a detecção precoce de erros. Este artigo tem como objectivo descrever um sistema estático de tipos para um subconjunto do Python (RPython). Acreditamos que a definição de um este sistema de inferência de tipos, como este, é um passo importante para a construção de um sistema de verificação formal de programas Python.

1 Introdução

A verificação formal de programas é, hoje, de reconhecida importância devido ao aumento da necessidade de certificar o *software* como fiável. Em especial, é importante certificar o *software* para os sistemas críticos e embebidos. Quando o desempenho destas aplicações não é crítico, a necessidade de segurança, correcção e rapidez de desenvolvimento justificam a utilização de linguagens de alto nível, como o Python.

Nos últimos trinta anos, os sistemas de tipos têm sido desenvolvidos e usados com sucesso em diferentes linguagens de programação. Um sistema de tipos, é um componente das linguagens tipificadas, que define um conjunto de regras que associam tipos aos objectos do programa. O uso de um sistema de tipos permite prevenir a ocorrência de determinados erros durante a execução do programa.

O Python [Ros95] é uma linguagem de programação de muito alto nível, orientada a objectos e dinamicamente tipificada. Possui uma sintaxe clara, que facilita a legibilidade do código e o desenvolvimento rápido de programas.

Neste trabalho apresentamos um sistema que permite a inferência estática de tipos em Python. Como esta linguagem possui algumas características que impossibilitam a inferência de tipos, na ausência de execução, consideramos um seu subconjunto designado RPython [AACM07]. O RPython foi definido informalmente no âmbito do projecto PyPy [Pro], cujo objectivo é a possibilidade de execução eficiente de Python e a construção de um compilador "Just-in-time". Para esta sub-linguagem é possível inferir tipos em tempo de compilação, uma vez que possui as seguintes características:

1. as variáveis têm tipo estático.

⋆ Trabalho parcialmente suportado pela Fundação de Ciência e Tecnologia e programa POSI, e pelo projecto RESCUE (PTDC/EIA/65862/2006)

2. os tipos complexos têm que ser homogéneos.
3. não possui características introspectivas nem reflexivas.
4. não permite o uso de métodos especiais (__*__), a definição de funções dentro de funções, a definição e uso de variáveis globais e apenas permite o uso de herança simples.

Existem alguns trabalhos relacionados com a inferência de tipos em Python [vS]. No entanto, nenhum deles procede à inferência de tipos, na ausência de execução, em Python, de modo formal.

2 Sistema de tipos

Um sistema de tipos define um conjunto de regras que associam tipos aos construtores de um programa.

A sintaxe abstracta do Python sobre a qual a inferência de tipos é efectuada é definida pela seguinte gramática, na qual não faremos distinção entre expressões e comandos:

$\texttt{e, } \bar{\texttt{e}} \texttt{ ::= n} \mid \texttt{l} \mid \texttt{x} \mid \texttt{(}e_1\texttt{, ..., }e_n\texttt{)}$ (*tuplos*) $\mid \texttt{[}e_1\texttt{,..., }e_n\texttt{]}$ (*listas*)
$\mid \{\bar{e_1}\texttt{:}e_1\texttt{,...,}\bar{e_n}\texttt{:}e_n\}$(*dicionários*) $\mid \texttt{x=e} \mid \texttt{e op e} \mid \texttt{e opc e} \mid \texttt{e opb e}$
$\mid \texttt{opu e} \mid \texttt{if e: e else e} \mid \texttt{e[n]} \mid \texttt{return} \mid \texttt{return e}$
$\mid \texttt{while e: e else e} \mid \texttt{def f(}x_1 \ldots x_n\texttt{):e} \mid \texttt{f(}e_1 \ldots e_n\texttt{)}$
$\mid \texttt{class c():[}e_1\texttt{,...,}e_n\texttt{]} \mid \texttt{c(}e_1\texttt{, ..., }e_n\texttt{)} \mid \texttt{e.m(}e_1\texttt{,..., }e_n\texttt{)} \mid \texttt{e.m}$

onde,

$\texttt{n} \in \{\text{int, float, long}\}$, $\texttt{l} \in$ constantes, $\texttt{x} \in$ nomes de variáveis
$\texttt{f} \in$ nomes de funções, $\texttt{c} \in$ nomes de classes, $\texttt{m} \in$ nomes de métodos

$\texttt{op::= +} \mid \texttt{-} \mid \texttt{*} \mid \texttt{<<} \mid \texttt{>>} \mid \texttt{|} \mid \texttt{|^} \mid \texttt{\&} \mid \texttt{/} \mid \texttt{\%} \mid \texttt{**} \mid \texttt{//}$
$\texttt{opc::= ==} \mid \texttt{!=} \mid \texttt{<} \mid \leq \mid > \mid \geq \mid \texttt{is} \mid \texttt{not is} \mid \texttt{in} \mid \texttt{not in}$
$\texttt{opb::= and} \mid \texttt{or}$
$\texttt{opu::= not} \mid \sim \mid \texttt{+} \mid \texttt{-}$

Consideremos o contexto local a uma classe, Ω, definido do seguinte modo:

$$\Omega \texttt{ ::= } \{\texttt{m}_0\texttt{::}\eta_0 \ldots \texttt{ m}_n\texttt{::}\eta_n\}$$

onde η_i se encontra definido abaixo.

O conjunto de tipos possíveis para a linguagem define-se pela seguinte gramática, onde `TVar` representa o conjunto das variáveis de tipo, τ e α os tipos monomórficos e η os tipos polimórficos:

$\tau \texttt{ ,}\alpha \texttt{ ::= eTop}$
$\mid \texttt{eInt} \mid \texttt{eFloat} \mid \texttt{eLong} \mid \texttt{eString} \mid \texttt{eBool} \mid \texttt{eNone} \mid \sigma \in \texttt{TVar}$
$\mid \texttt{eTuple(}\tau_1 \ldots \tau_n\texttt{)} \mid \texttt{eList(}\tau\texttt{)} \mid \texttt{eDict(}\tau\texttt{)} \mid \texttt{eArrow([}\tau_1 \ldots \tau_n\texttt{],}\alpha\texttt{)}$
$\mid \texttt{eClass(c,}\Omega\texttt{)} \mid \texttt{eCCla(l, [}c_1\texttt{,..., }c_n\texttt{])} \mid \texttt{eCv(l, [}\tau_1\texttt{,...,}\tau_n\texttt{])}$
$\eta \texttt{ ::= } \tau$
$\mid \texttt{eAll([}\sigma_1\texttt{,...,}\sigma_n\texttt{], eArrow([}\tau_1 \ldots \tau_n\texttt{],}\alpha\texttt{))}$

2.1 Regras de inferência

Ao conjunto das atribuições de tipo a variáveis ou funções, distintas, chamamos contexto, e representamos por Γ. O contexto é global durante todo o processo de inferência. A definição deste conjunto, onde $t_i \in x, f$, é a seguinte:

$$\Gamma ::= \{t_0 :: \eta_0, ..., t_n :: \eta_n\}$$

Dado um contexto Γ, um construtor $\mathtt{e}$ e um tipo τ, $\Gamma \vdash \mathtt{e} ::\tau$ significa que considerando o contexto Γ é possível deduzir que o construtor $\mathtt{e}$ tem tipo τ.

De seguida, vamos definir algumas das regras de inferência para o sistema de tipos.

$$\frac{\Gamma \vdash \mathtt{x} :: \tau,\ \mathtt{se}\ (\mathtt{x} :: \tau) \in \Gamma \quad \Gamma \vdash \mathtt{e_i} :: \tau_\mathtt{i}\ \mathtt{1} \leq \mathtt{i} \leq \mathtt{n}}{\Gamma \vdash (\mathtt{e_1}, ..., \mathtt{e_n}) :: \mathtt{eTuple}\ ([\tau_1, ..., \tau_\mathtt{n}])}\ (\textsc{Tuplo})$$

$$\frac{\Gamma \vdash \mathtt{e_i} :: \tau\ \mathtt{1} \leq \mathtt{i} \leq \mathtt{n}}{\Gamma \vdash [\mathtt{e_1}, ..., \mathtt{e_n}] :: \mathtt{eList}\ (\tau)}\ (\textsc{Lst})$$

$$\frac{\Gamma \vdash \bar{\mathtt{e_i}} :: \alpha_\mathtt{i}\ \mathtt{hashable}(\alpha_\mathtt{i}) \quad \Gamma \vdash \mathtt{e_i} :: \tau\ \mathtt{1} \leq \mathtt{i} \leq \mathtt{n}}{\{\bar{\mathtt{e_1}} : \mathtt{e_1}, ..., \bar{\mathtt{e_n}} : \mathtt{e_n}\} :: \mathtt{eDict}\ (\tau)}\ (\textsc{Dic})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \tau_1\ \Gamma \vdash \mathtt{x} :: \tau_2 \quad \tau_1 <: \tau_2\ \mathtt{ou}\ \tau_2 <: \tau_1}{\Gamma \vdash \mathtt{x} = \mathtt{e} :: \mathtt{eNone}}\ (\textsc{Atr})$$

$$\frac{\Gamma \vdash \mathtt{e_1} :: \tau_1\ \ \Gamma \vdash \mathtt{e_2} :: \tau_2\ \ \tau_1 <: \tau_2}{\Gamma \vdash \mathtt{e_1}\ +\ \mathtt{e_2} :: \tau_2}\ (\textsc{Opb1})$$

$$\frac{\Gamma \vdash \mathtt{e_1} :: \tau_1\ \ \Gamma \vdash \mathtt{e_2} :: \tau_2\ \ \tau_2 <: \tau_1}{\Gamma \vdash \mathtt{e_1}\ +\ \mathtt{e_2} :: \tau_1}\ (\textsc{Opb2})$$

$$\tau_1, \tau_2 \in \{eInt,\ eFloat,\ eLong,\ eString,\ eTuple(\alpha),\ eList(\alpha)\}$$

$$\frac{\Gamma \vdash \mathtt{e_1} :: \tau_1\ \Gamma \vdash \mathtt{e_2} :: \tau_2 \quad \tau_1 <: \tau_2\ \mathtt{ou}\ \tau_2 <: \tau_1}{\Gamma \vdash \mathtt{e_1}\ \mathtt{opc}\ \mathtt{e_2} :: \mathtt{eBool}}\ (\textsc{Opc1})$$

$$\frac{\Gamma \vdash \mathtt{e_1} :: \mathtt{eBool}\ \ \Gamma \vdash \mathtt{e_2} :: \mathtt{eBool}}{\Gamma \vdash \mathtt{e_1}\ \mathtt{opb}\ \mathtt{e_2} :: \mathtt{eBool}}\ (\textsc{Opbool})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \mathtt{eList}(\tau)\ \mathtt{i} :: \mathtt{eInt}}{\Gamma \vdash \mathtt{e[i]} :: \tau}\ (\textsc{Alst1})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \mathtt{eList}(\tau)\ \mathtt{n} :: \mathtt{eInt}\ \mathtt{m} :: \mathtt{eInt}}{\Gamma \vdash \mathtt{e[n:m]} :: \mathtt{eList}(\tau)}\ (\textsc{Alst2})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \mathtt{eDict}(\tau) \quad \Gamma \vdash \mathtt{i} :: \alpha\ \mathtt{hashable}(\alpha)}{\Gamma \vdash \mathtt{e[i]} :: \tau}\ (\textsc{Adic1})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \mathtt{eDict}(\mathtt{eNone}) \quad \Gamma \vdash \mathtt{i} :: \alpha\ \mathtt{hashable}(\alpha)}{\Gamma \vdash \mathtt{e[i]} :: \mathtt{eTop}}\ (\textsc{Adic2})$$

$$\Gamma \vdash \mathtt{return} :: \mathtt{eNone}\ (\textsc{Return1})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \tau}{\Gamma \vdash \mathtt{return\ e} :: \tau}\ (\textsc{Return2})$$

$$\frac{\Gamma \vdash \mathtt{e_0} :: \mathtt{eBool}\ \Gamma \vdash \mathtt{e_1} :: \tau \quad \Gamma \vdash \mathtt{e_2} :: \alpha \quad \tau <: \alpha}{\Gamma \vdash \mathtt{if\ e_0 :\ e_1\ else\ e_2} :: \alpha}\ (\textsc{Cond1})$$

$$\frac{\Gamma \vdash \mathtt{e_0} :: \mathtt{eBool}\ \Gamma \vdash \mathtt{e_1} :: \tau \quad \Gamma \vdash \mathtt{e_2} :: \alpha \quad \alpha <: \tau}{\Gamma \vdash \mathtt{if\ e_0 :\ e_1\ else\ e_2} :: \tau}\ (\textsc{Cond2})$$

$$\frac{\bar{\Gamma} = \{\mathtt{x_i} :: \tau_\mathtt{i}\}\ \mathtt{1} \leq \mathtt{i} \leq \mathtt{n}\ \bar{\Gamma} \cup \Gamma' \vdash \mathtt{e} :: \alpha}{\Gamma'' \vdash \mathtt{def\ f}\ (\mathtt{x_1}, ..., \mathtt{x_n}) : \mathtt{e} :: \mathtt{eArrow}([\tau_1, ..., \tau_\mathtt{n}], \alpha)}\ (\textsc{DefFunc})$$

$\Gamma'' = \Gamma \cup f :: eArrow([\tau_1, ..., \tau_n], \alpha)$

$$\frac{\Gamma \vdash \mathtt{f} :: \mathtt{eArrow}([\tau_1, ..., \tau_n], \alpha)\ \Gamma \vdash \bar{\mathtt{e}}_i :: \alpha_i\ \alpha_i\ <:\ \tau_i\ 1 \leq i \leq n}{\Gamma \vdash \mathtt{f}(\bar{e_1}, ..., \bar{e_n})\ ::\ \alpha}\ (\textsc{Aplicação})$$

$$\frac{\bar{\Gamma} = \{\ \mathtt{e}_i :: \tau_i\} \ 1 \leq i \leq n\ \bar{\Gamma} \cup \Gamma' \vdash \mathtt{e} :: \alpha}{\Gamma'' \vdash \mathtt{def\ f}(\mathtt{e}_1, ..., \mathtt{e}_n) : \mathtt{e}\ ::\ \mathtt{eAll}([\tau_i \in \mathtt{TVar}], \mathtt{eArrow}([\tau_i], \alpha))}\ (\textsc{Generalização})$$

$$\Gamma'' = \Gamma \cup f :: eAll([\sigma_1, ..., \sigma_n], eArrow([\tau_1, ..., \tau_n], \alpha))$$

$$\frac{\Gamma' \vdash \mathtt{e}_i :: \eta_i\ \ 1 \leq i \leq n}{\Gamma \vdash \mathtt{class\ c}() : [\mathtt{e}_1, ..., \mathtt{e}_n]\ ::\ \mathtt{eClass}(\mathtt{c}, \{\mathtt{m}_1 :: \eta_1, ..., \mathtt{m}_n :: \eta_n\})}\ (\textsc{DefCla})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{c} :: \mathtt{eClass}(\mathtt{c}, \Omega) \\ \Gamma,\ \Omega \vdash \mathtt{\_\_init\_\_}(\mathtt{e}_1, ..., \mathtt{e}_n) :: \mathtt{eNone}()\end{array}}{\Gamma \vdash \mathtt{c}(\mathtt{e}_1, ..., \mathtt{e}_n) :: \mathtt{eClass}(\mathtt{c}, \Omega)}\ (\textsc{Inst1}) \qquad \frac{\begin{array}{c}\Gamma \vdash \mathtt{c}(\mathtt{e}_1, ..., \mathtt{e}_n) :: \mathtt{eClass}(\mathtt{c}, \Omega) \\ \Omega \vdash \mathtt{m}(\tau_1, ..., \tau_n) :: \eta \\ \Gamma \vdash \bar{\mathtt{e}_i} :: \alpha_i\ \alpha_i <: \tau_i\ 1 \leq i \leq n\end{array}}{\Gamma \vdash \mathtt{c}(\mathtt{e}_1, ..., \mathtt{e}_n).\mathtt{m}(\bar{e_1}, ..., \bar{e_n}) :: \eta}\ (\textsc{Acm1})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{c} :: \mathtt{eClass}(\mathtt{c}, \Omega) \\ \Gamma,\ \Omega \vdash \mathtt{\_\_init\_\_}(\mathtt{e}_1, ..., \mathtt{e}_n) :: \mathtt{eClass}(\mathtt{c}, \Omega)\end{array}}{\Gamma \vdash \mathtt{c}(\mathtt{e}_1, ..., \mathtt{e}_n) :: \mathtt{eClass}(\mathtt{c}, \Omega)}\ (\textsc{Inst2}) \qquad \frac{\begin{array}{c}\Gamma \vdash \mathtt{c}(\mathtt{e}_1, ..., \mathtt{e}_n) :: \mathtt{eClass}(\mathtt{c}, \Omega) \\ \Omega \vdash \mathtt{m} :: \eta\end{array}}{\Gamma \vdash \mathtt{c}(\mathtt{e}_1, ..., \mathtt{e}_n).\mathtt{m} :: \eta}\ (\textsc{Acm2})$$

3 Conclusão

O sistema de inferência apresentado foi implementado em Python e está apresentado em pormenor na tese de mestrado *Inferência de tipos em Python* [Mai].

Actualmente a certificação de software, como correcto e seguro, é de extrema importância, especialmente para sistemas críticos e embebidos. Muitas das aplicações usadas nestes sistemas são desenvolvidas em linguagens de alto-nível, como o Python. Desejamos encadear o sistema de inferência de tipos aqui apresentado com uma ferramenta de produção de obrigações de prova. Assim, o desenvolvimento deste sistema estático de inferência de tipos foi apenas o primeiro passo para um projecto futuro que implemente a certificação estática de programas em Python.

Referências

[AACM07] Davide Ancona, Massimo Ancona, Antonio Cuni, and Nicholas D. Matsakis. Rpython: a step towards reconciling dynamically and statically typed oo languages. In *DLS '07: Proceedings of the 2007 symposium on Dynamic languages*, pages 53–64, New York, NY, USA, 2007. ACM.

[Mai] Eva Maia. Inferência de tipos em python.

[Pro] PyPy Project. Pypy: flexible and fast python implementation.

[Ros95] Guido Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[vS] Anton van Straaten. Type inference for python. *Lambda the Ultimate The Programming Languages Weblog.*

Reasoning about time-critical reactive systems: A case-study

André M. Rodrigues da Silva

DI - CCTC, Universidade do Minho
Campus de Gualtar
4710-057 – Braga – Portugal
a47408@alunos.uminho.pt

Abstract. Nowadays, there is a lot of technology that must not fail at any circumstances. A car airbag is a simple reactive system that must work every time the car crashes. In order to model problems like this, there are some model-checking applications that can help us do the job. In this paper a well known case study will be modeled and verified in Uppaal and some variants of the problem will be documented. At the end the problem will be generalized in order to make it easy to apply to other problems with different values.

Keywords: Uppaal, timed automata, reactive system, case-study

1 Introduction

This paper presents a case-study on modeling and verification of time-critical systems using timed automata [2] and the Uppaal model checker [4]. The case study revisits a well-known problem on multimedia design: the specification and verification of media streams subject to a number of quality of service constraints, namely end-to-end latency and throughput. The proposed solution improves, in what concerns generality, a previous attempt documented in [5]

The full version of this paper can be found at `http://tinyurl.com/paper55`.

2 Case study: The media stream channel revisited

The problem chosen as a case-study for this paper was a media stream channel. This has three elements: the *Source* that emits messages, the *Sink* that receives and processes them, and the *Channel* that establishes the connection between the *Source* and the *Sink*.

These elements are obliged to the following requirements:

- *Source* emits a message every 50ms;
- *Channel* takes between 80ms and 90ms to deliver the message;
- *Channel* may loose messages, but no more than 20% of them;
- A message is considered lost if it does not arrive at the *Sink* within 90ms;

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 519–522

– *Sink* takes 5ms to process each received message;
– An error should be generated if less than 15 messages per second arrive to the *Sink*.

These requirements should be modeled in Uppaal, first by simulation in order to get a feeling about its behavior and then tested through the verification of suitable queries to check if the envisaged requirements are indeed enforced.

3 Problems and Solutions

One of the problems we have faced was to find a way to avoid the absence of control over the messages. Actually, each message is sent by the *Source* every 50ms, and the *Channel* must have a way to simulate the delay of the communication of that message. Moreover, it must be possible to have more than one message in delay at the same time.

Fortunately, it was realized that two processes alone were enough to solve the problem, because of the *Source* frequency and the *Channel* latency values — this was the crucial observation in this case-study. Such two processes must have the possibility of being reused, inasmuch as when one of them starts processing one message, it will be busy for at most 90ms, and, as in this interval another message may be generated, there must be another process ready to synchronize.

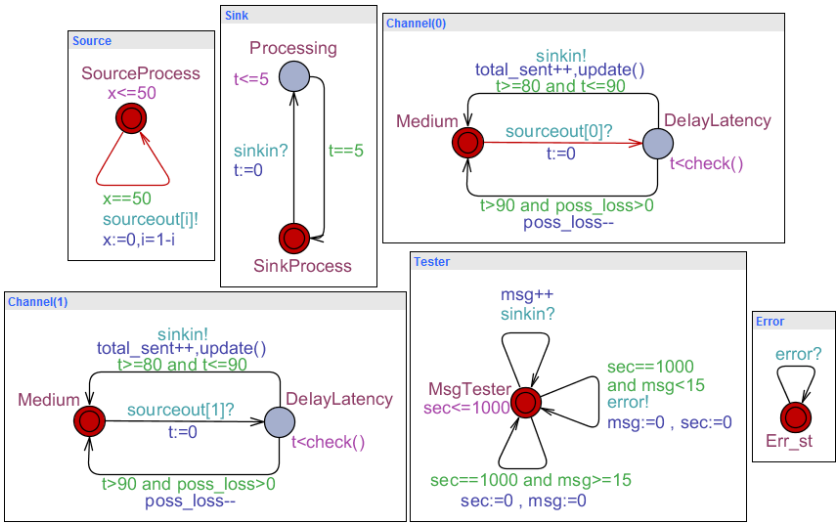


Fig. 1. A possible solution.

In order to make this process reusable, the initial state is always waiting for a message. On its arrival the process resets the clock and goes to a state where it waits for a time t between 80ms and 91ms as shown in Fig. 1. Invariant

$t < check()$ attached to this state caters for the possibility of losing messages. Note that the transitions must have a parameter to identify which *Channel* process will receive the message.

To make sure there is an error if less than 15 messages are processed by the *Sink*, a *Tester* process was created to count the number of successful sinkin synchronizations per second. To do this, the latter increments a variable each time a message is sent and resets the clocks and the counter every seconds, sending an error to the *Error* process if the number of messages is less than 15.

In order to make sure that *Channel* does not loose more than 20% of the messages, the number of total messages and the number of messages lost should be known. But we can't have unbounded integers with this information, because when we try to verify a property, the proof tree would grow till the maximum integer allowed is reached or till the memory runs out. To fix this problem, we resort to an easy calculation: saying that *Channel* may loose messages, but no more than 20% is the same as saying that in every 5 messages, 1 may be lost. So, instead of counting all the messages, we can decide to increment the counter until it reaches 5, and then reset it, incrementing one possibility of loss.

The counter that stores the number of possible losses at any moment still remains in model and must be delimited. Actually it makes no sense for a media channel to have a huge number of possible losses, even because the *Sink* will emit an error if it receives less than 15 messages per second. So it is acceptable if we introduce a little error here, that in theory will decrease the percentage fixed in the requirements, meaning that less losses are allowed. To implement this, the function *update()*, only tolerates the counter *poss_loss* to be incremented until it reaches a MAX value, globally defined. This value will maximize the number of states considered in the proof tree; and the smaller this number, the faster the proof will be.

The check function mentioned above is defined as follows

```
int check()
{ if (poss_loss>0) return 91;
   return 90; }
```

and will enable or disable the transition of loss depending on the value of *poss_loss*.
A property selected for verification was the most obvious:

```
A[] not deadlock
```

i.e. no state reaches a deadlock configuration. Simple as it was, its verification was a considerable challenge, but after the replacement of the unbounded counters, the property was verified.

4 Concluding

The final model, depicted in Fig. 1, can be easily generalized to other media stream channels, with different latencies and frequencies. In order to do that, the frequency of the source , the maximum latency of the channel and the interval of latency should be declared as integer constants respecting the following rules:

```
const int source_freq=10;   //must be greater than Sink process time
const int max_latency=150;
const int latency_interv=5;   //must be <= source_freq
const int N=max_latency/source_freq+1;
```

With the values above, 15 channel processes are created by the Simulator, as the value of constant N is computed to guarantee deadlock avoidance. Restrictions at the source frequency and the latency interval are due to the *Sink* processing time, because there is only one instance of it. Note that the latency interval must be less than the source frequency. Otherwise one process could be trying to send a message to the *Sink* while another message was still being processed. Note that the *Source* must synchronize with more than two processes, so the index of the transition must be incremented until it reaches $N - 1$, being then restarted.

The case study discussed in this paper, set as an exercise on the practical use of Uppaal, illustrated how this sort of tool-supported formal methods can give a precious help to the design of real-time, complex systems. The final solution not only followed a different strategy, but also represents a more general solution to the media stream problem than the one proposed in [5].

The Uppaal simulator is a excellent help when creating a model, because it allows debugging even before the specification is mature and complete enough to be model-checked. Note, however, its role is always limited to that of an animator of the intended behaviour. In this case-study, for example, the verifier found a problem that could not be handled by the the simulator, the latter being unable to pursue exhaustive checking. However, the error found could have been a lot easier to debug if the model-checker had issued a warning, informing that the variable was not defined as a bounded integer and was being incremented.

In concluding, it is a fact that model checking [3] is, at present, a mature collection of techniques and practical tools for automated debugging of complex reactive systems [1]. Actually, as this small case-study may have helped to illustrate, it is no more a theoretical oddity, but, on contrary, its relevance for the working software engineering cannot be understated.

Acknowledgments. I would like to thank to Luís Soares Barbosa for all the support and encouragement he gave me, because without it, I wouldn't have written this short article.

References

1. L. Aceto, A. Ingólfdótir, K. G. Larsen, and J. Srba. *Reactive Systems: Modelling, specification and verification*. Cambridge University Press, 2007.
2. R. Alur. Timed automata. *Theoretical Computer Science*, 126:183–235, 1999.
3. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4. J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal - a tool suite for automatic verification of real-time systems, 1996.
5. H. Bowman, G. P. Faconti, and M. Massink. Specification and verification of media constraints using upaal. In *Design, Specification and Verification of Interactive Systems'98, Proceedings of the Fifth International Eurographics Workshop, June 3-5, 1998, Abingdon, United Kingdom*, volume 1, pages 261–277. Springer, 1998.

Gestão e Tratamento de Informação

A Search Log Analysis of a Portuguese Web Search Engine

Miguel Costa[1,2] and Mário J. Silva[2]
(miguel.costa@fccn.pt, mjs@di.fc.ul.pt)

[1] Foundation for National Scientific Computing, Lisbon, Portugal
[2] University of Lisbon, Faculty of Sciences, LaSIGE, Lisbon, Portugal

Abstract. We present a characterization of the information-seeking behavior of the users of a Portuguese web search engine, based on the analysis of its logs. We obtained detailed statistics about the users' sessions, queries, terms and searched topics over a period of two years. The results show that the users prefer fast and short sessions, composed of short queries and few clicks. The trend is towards a reduction of the number of interactions with the web search engine. We also discuss the specificities and interests of the Portuguese users and their implications on the development of better adapted web search engines.

1 Introduction

Web search engines are one of the most used systems on the Internet. Commercial web search engines, like Google and Yahoo!, receive hundreds of millions of queries per day [1]. Their goal is to satisfy the users' information needs as well as possible. Hence, it is necessary to understand what and how users search, what they expected and the difficulties they face when seeking information.

User studies analyze user behavior through several methods, some quantitative, such as surveys and log mining [1, 2], and others qualitative, such as observations and think-aloud protocols [3, 4]. Qualitative analysis can provide valuable insights on the usability of systems and user satisfaction. However, the time spent experimenting with participants and the costs of acquiring specialized equipments, often lead researchers to reduce the users sample to a size smaller than required to obtain statistically significant results. Another problem of qualitative methods is their intrusiveness in the search process. Just the fact that the users are aware of being observed can affect their normal behavior.

On the other hand, search logs capture a large and varied amount of interactions between users and search engines. This large number of interactions is less susceptible to bias and enables identifying stronger relationships between the data. Additionally, analyses of search logs can be cheaper and non intrusive. Previous studies based on search logs show that there are differences between users from different world regions. The users' behavior reflects their distinct language, vocabulary and cultural bindings. For instance, Korean users submit on

[1] see blog.nielsen.com/nielsenwire/online_mobile/nielsen-reports-march-2010-u-s-search-rankings

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 525–536

average queries with one term less than U.S. and European users, because their words are often compound nouns [5]. European users also search slightly differently than the U.S. users [6]. For instance, Europeans search more about people and places, while the U.S. users are more focused on e-commerce [1]. Hence, we considered it important to study the specificities of the Portuguese web search engine users and the degree to which the searching technology applied to other countries can be adopted.

This study draws the first profile of Portuguese users. It is based on the 2003 and 2004 search logs from the Tumba! web search engine [7]. As far as we know, this is also the first study about users whose native language is Portuguese, considered the fifth language with more users on the Internet [2]. This profile is slightly outdated since the logs are six years old. Web search engine companies have become increasingly reluctant of releasing their logs because of privacy concerns [8], so logs are now scarce and outdated. However, these logs enable us to directly compare the results with similar studies from the same period. This study is also a baseline for new research over more current logs and might contribute to the development of better adapted web search engines. Examples include optimizing their performance [9] or designing better web interfaces [10].

This paper is organized as follows. In Section 2, we cover the related work. In Section 3, we describe the logs dataset from which we based our study. The methodology of analysis is explained in Section 4 and the results are detailed in Section 5. Section 6 finalizes with the discussion of results and conclusions.

2 Related Work

Several studies performed quantitative analysis of search logs in the past, with the goal of understanding how web search engines were used. A common observation across these studies is that most users conduct short sessions with only one or two queries, composed by one or two terms each [1]. When users submit more than one query, they tend to refine the next query by changing one term at a time. Most users only see the first search engine results page (SERP) and rarely use advanced search operators. These discoveries imply that the use of web search engines is different from traditional IR systems, which receive queries three to seven times longer [11]. Queries for special topics (e.g. sex) and multimedia formats (e.g. images) are also longer [2].

A comparison by Spink et al. of the searching behaviors of users from the U.S. and Europe, pointed out a few differences [6]. Statistics for U.S. and European users were collected from the Excite and FAST web search engines, respectively. FAST users were mostly from Germany and Norway. U.S. users submitted on average more terms per query (2.6 vs. 2.3), but less queries per session (2.3 vs. 2.9) and accessed less SERPs (1.7 vs. 2.2). Longer queries may produce better results and this may explain why the U.S. users explored less SERPs. U.S. and European users also searched topics differently. U.S. users searched more about *Commerce, Travel, Employment or Economy*, while European users searched

[2] see `http://www.internetworldstats.com/stats7.htm`

more about *People, Places or Things*. However, Spink et al. only classified the English language queries for the supposed German and Norwegian users, which could have skewed the results.

The evolution of behaviors throughout time hardly changed. The longitudinal study of Spink et al. over Excite's users of 1997, 1999 and 2001, showed that the only significant difference was an increase, from 28.6% in 1997 to 50.5% in 2001, in the number of queries where only one SERP was viewed [12]. Another finding was that search topics shifted from entertainment and sex to commerce and people. The authors stated that e-commerce queries coincided with changes on the information distribution of the web. According to Lawrence and Giles, in 1999 about 83% of the web servers contained commercial content [13].

Jansen's analysis of the Altavista logs of 1998 and 2002, indicated that users evolved to longer queries, longer sessions, more modified queries and less results seen [14]. It seems that users became more persistent when searching for information. However, the results from 1998 could have been affected by the inactivity timeout of 5 minutes selected to delimit sessions, which is shorter than in posterior studies. Jansen and Spink analyzed users from FAST in 2001 and 2002 [15]. They detected a move toward a great simplicity in searching. Single query sessions and single term queries increased. The percentage of users refining queries decreased. On the other hand, users saw more SERPs. The frequency of topics searched by the European users also changed. There was a large increase from 22.5% to 41.5% of queries searching about *People, Places or Things*. On the other hand, there was a decline of more than 5% for queries searching about *Computers or Internet* and *Sex or Pornography*.

There were other important discoveries. Beitzel et al. showed that the volume of queries varies along the hours of the day and the days of the week [16]. The topics searched are also more or less popular at different times of the day. Hölscher and Strube discovered that expert users submit more complex queries and have more flexible searching strategies than the newbies [4]. Ozmutlu et al. pointed out that users may search for multiple topics in a single session [17].

3 Tumba!'s Logs Dataset

Tumba! was a search engine for the Portuguese web, which was available as a public service from 2002 to 2006 [7]. At the time, the Portuguese web was considered the subset of web pages satisfying one of the following conditions: (1) hosted on a site under a .PT domain; (2) hosted on a site under other domain (except .BR), written in Portuguese and with at least one incoming link from a web page hosted under a .PT domain. The interaction with the users and the layout of the results was similar to other web search engines, such as Google.

Our analysis is based on the Tumba!'s logs, covering two full years of search interactions, 2003 and 2004. By interactions, we mean all queries and clicks submitted by the users and recorded by the web search engine. This large time range has several advantages. First, we can see how the users evolved over the years. Second, it is less likely to be affected by ephemeral trends. Third, we can identify seasonal search patterns, for instance, during the Christmas season.

The logs follow the Apache Common Log Format (see `http://httpd.apache.org/docs/2.0/logs.html`). Each entry corresponds to an interaction with the search engine in the form of a HTTP request. It contains the user's IP address and the user's session identifier (id). However, Tumba! did not register the session id in the log. Each entry contains also a timestamp indicating when the interaction occurred, the HTTP request line that came from the client and two parameters of the data sent back by the server, which are the HTTP response' status code and size. Figure 1 presents three entries of this log, which are purely illustrative.

```
213.22.91.10 - [03/Feb/2004:23:15:27 +0000] "GET ?q=lisbon&lang=pt HTTP/1.1" 200 19978
213.22.91.10 - [03/Feb/2004:23:15:31 +0000] "GET ?q=lisbon&lang=pt&start=10 HTTP/1.1" 200 21419
213.22.91.10 - [03/Feb/2004:23:15:33 +0000] "GET ?q=lisbon&lang=pt&start=10&
  click=pt.wikipedia.org/wiki/Lisbon&rank=12 HTTP/1.1" 200 18409
```

Fig. 1: Log entries format.

We never used the log data to match a real identity. However, we had to check that these logs did indeed correspond to Portuguese users. We counted 90% of Tumba!'s users with IP addresses assigned to Portugal and near 98% of the interactions were submitted through the Portuguese language interface. This strongly indicates that the users were mostly Portuguese.

4 Methodology

The analysis focused on four dimensions: sessions, queries, terms and clicks. We define them in the following way:

- A *session* is a set of interactions that belong to the same user when attempting to satisfy one information need. The session is the level of analysis in determining the success or failure of a search. It is composed by one or more queries and zero or more clicks.
- A *query* is a search request composed by a set of terms. We define an *initial query* as the first query submitted in a session, while all the following queries are defined as *subsequent*. An *identical query* is a query with exactly the same terms as the previous one and submitted in the same session. A *unique query* corresponds to one query regardless of the number of times it was logged. The set of unique queries is the set of query variations. An *advanced query* is a query with at least one advanced operator.
- A *term* is a series of characters bounded by white spaces, such as words, numbers, abbreviations, URLs, symbols or combinations between them. There are also advanced search operators, but they are not counted as terms. We define a *unique term* as one term on the dataset regardless of the number of times it was logged. The set of unique terms is the submitted lexicon.
- A *click* in this context refers to the following of a hyperlink in a SERP to immediately view a query result (i.e. web page).

Next, we briefly present the methods used on the search log analysis.

4.1 Log preparation

Abnormal sessions and queries could skew the results of the study. Thus, we started by preparing the log fields for analysis through a series of data cleansing steps. All incomplete entries, empty queries and sessions without any query were discarded. Internal queries submitted by the Tumba! watchdog and the sessions with more than 100 queries were also excluded. Sessions with many queries were likely to come from web crawlers and we were only interested in the queries submitted by users. This cutoff value of 100 was also used in some other studies, thus enabling a more direct comparison with our results [15, 14]. The queries that resulted from navigation clicks to see another SERP were not counted as a new query. These are the same queries parameterized to show more results.

All terms were normalized to lowercase since capitalization was ignored. Extra white spaces were removed and since the search engine did not perform stemming, all variations of a query term were considered as different terms. The set of query terms also includes punctuation, misspellings and mistakes.

4.2 Session delimitation

Previous studies used the users' IP address and/or session identifier (id) to delimit sessions. However, the Tumba! logs did not capture the session id. Hence, we cannot tell if the requests with the same IP came from different computers behind the same proxy server.

Most studies also used a time interval t of inactivity to delimit sessions. Two consecutive interactions are included on different sessions if they have an inactivity between them of at least t. This gap serves to separate two information needs of the same user, asked at different times. Without this gap, we could have sessions of several days, which would hardly represent the reality. Studies diverge on the choice of this interval, from 5 minutes [18] to 30 minutes [19], while others argue that no time boundary is effective in segmenting sessions [20]. We selected the 30 minute interval, since 95% of the sessions are shorter and because it is the session default timeout on most web applications. This interval also produced results close to the ones of SVM classifiers used for delimiting sessions [21].

5 Log Analysis

The logged interactions and their parameters were statistically accounted. The users of the Portuguese web search engine Tumba! performed 254,728 and 133,827 searching sessions in 2003 and 2004, respectively. Analyzing the averages, the users submitted 2.94 queries per session in 2003 and 2.49 in 2004. In these two years, the average number of query terms was around 2.2, with nearly 7 characters per term. The users saw 1.4 SERPs per query and clicked around 0.7 times on their hyperlinks to view a result. This means that for each query, the users saw mostly the first and sometimes the next SERP, where they clicked at most once. Table 1 shows these general statistics. The results remained almost constant during the two years, except for a decrease in the number of queries per session. Next, we will detail our analysis and explain the remaining results.

Dataset	1 year-2003	1 year-2004
Sessions	254,728	133,827
Queries	749,914	333,871
Terms	1,630,392	738,576
SERPs	1,087,369	474,157
Clicks	584,161	240,961
Queries per Session	2.94	2.49
Terms per Query	2.17	2.21
SERPs per Query	1.45	1.42
Clicks per Query	0.78	0.72
Characters per Term	6.99	6.80
Initial Queries	33.97%	40.08%
Subsequent Queries	66.03%	59.92%
- Modified	32.80%	33.48%
- Identical	29.35%	32.71%
- Terms Swapped	0.26%	0.29%
- New	37.59%	33.52%
Unique Queries	44.03%	48.52%
Unique Terms	8.00%	10.33%
Queries never repeated	30.02%	34.04%
Terms never repeated	3.72%	4.77%

Table 1: General statistics.

Session duration	year 2003 % sessions	year 2004 % sessions	Δ
$[0, 1[$	43.18%	53.31%	+10.13%
$[1, 5[$	25.89%	21.67%	-4.22%
$[5, 10[$	10.69%	8.91%	-1.78%
$[10, 15[$	5.88%	4.80%	-1.08%
$[15, 30[$	8.89%	7.29%	-1.60%
$[30, 60[$	4.47%	3.33%	-1.14%
$[60, 120[$	0.93%	0.64%	-0.29%
$[120, 180[$	0.07%	0.04%	-0.03%
$[180, 240[$	0.01%	0.01%	0.00%
$[240, \infty[$	0.00%	0.00%	0.00%

Table 2: Session duration (minutes).

# queries	year 2003 % sessions	year 2004 % sessions	Δ
1	40.73%	49.52%	+8.79%
2	22.10%	21.10%	-1.00%
3	12.71%	10.86%	-1.85%
4	7.76%	6.09%	-1.67%
5	4.97%	3.84%	-1.13%
6	3.24%	2.43%	-0.81%
7	2.24%	1.61%	-0.63%
8	1.50%	1.17%	-0.33%
9	1.09%	0.78%	-0.31%
$\geq$10	3.67%	2.61%	-1.06%

Table 3: Number of queries per session.

5.1 Session Level Analysis

Session duration The duration of a session is measured as the time between the first query submitted until the last time the user interacted with the search engine. We ignore if the user spent more session time viewing web pages clicked from the SERP or used part of the time doing parallel tasks [17].

We can see on Table 2 that sessions ended quickly and the tendency is to end even faster. There was an increase, from 43.18% in 2003 to 53.31% in 2004, of the sessions with less than 1 minute. The average duration of the sessions also decreased, from 6 minutes and 31 seconds in 2003 to exactly 5 minutes in 2004. Around 80% of the sessions lasted less than 10 minutes and only less than 1% had a duration longer than one hour.

Query distribution Table 3 shows that the majority of the users did not go beyond their second query. Information retrieval is an iterative process, but the users hardly iterated. Around 90% of the sessions had up to 5 queries and only less than 4% had 10 or more queries. This last number can represent highly motivated users searching for special topics (e.g. sex) [2]. The results also show that there was an increase of almost 9% on sessions with only one query from 2003 to 2004. This is the main reason why the averages of the queries per session and the session duration decreased between the two years.

5.2 Query Level Analysis

Modified queries Sequences of queries are sometimes a way for users to refine or reformulate the search in a trial and error approach. A modified query is

# terms changed	year 2003 % modified queries	year 2004 % modified queries	Δ
≤-5	0.32%	0.40%	+0.08%
-4	0.48%	0.55%	+0.07%
-3	1.48%	1.71%	+0.23%
-2	5.01%	5.41%	+0.40%
-1	15.58%	15.77%	+0.19%
0	30.23%	28.97%	-1.26%
+1	36.52%	35.56%	-0.96%
+2	7.39%	8.23%	+0.84%
+3	1.97%	2.18%	+0.21%
+4	0.64%	0.79%	+0.15%
≥+5	0.37%	0.44%	+0.07%

Table 4: Number of terms changed per modified query.

defined as a subsequent query pertaining to the same information need and it is assumed that two queries have the same information need if they share at least one term. We ignored the stopwords (too common terms) in this analysis. Thus, a modified query could be a specialization of the query (adding terms), a generalization (removing terms) or both at the same time.

We counted 32.80% in 2003 and 33.48% in 2004 of modified queries from all subsequent queries (see Table 1). Looking to Table 4, we see that more than 80% of the modified queries are the result of a zero or one change on the number of terms. A zero length change means that the users modified some terms, but their number remained the same. Users tend to add more terms in the modified queries rather than to remove them. We counted around 47% versus 23%. As other users, Tumba!'s users tend to go from broad to narrow queries [11, 22, 18].

Identical and New queries Sometimes the users repeat queries. This can happen for a variety of reasons, such as a refresh of the SERP, a back-button click or the submission of the same query more than once due to a network or search engine delay. We counted 29.35% in 2003 and 32.71% in 2004 of identical queries (see Table 1), where each query is exactly the same as the previous one made in the same session. We also counted the subsequent queries with the same terms, but written in a different order. For instance, a query *Lisbon Portugal* followed by a query *Portugal Lisbon*. Only a small number of subsequent queries, 0.26% in 2003 and 0.29% in 2004, had the order of the terms swapped. Besides the modified and identical queries, the users also submitted in the same session, 37.59% in 2003 and 33.52% in 2004, of subsequent queries with only new terms (see Table 1). This indicates that at most, around of one third of the subsequent queries are the result of a new information need.

Advanced queries An advanced query is a query with at least one advanced operator. In Tumba!, the users could use three advanced operators: NOT, to exclude all results with a term in their text (e.g. *-Lisbon*); PHRASE, to match all results with a phrase in their text (e.g. *"cities of Portugal"*); SITE, to match all results from a domain name (e.g. *site:wikipedia.org*).

Table 5 contains the percentages of advanced queries. It shows that only, 12.79% in 2003 and 11.40% in 2004, of the queries included advanced operators.

advanced operator	year 2003		year 2004		Δ
	% adv. queries	% total queries	% adv. queries	% total queries	% adv. queries
NOT	2.91%	0.37%	3.60%	0.41%	+0.69%
PHRASE	43.15%	5.52%	49.93%	5.69%	+6.78%
SITE	53.95%	6.90%	46.46%	5.30%	-7.49%
total	100.00%	12.79%	100.00%	11.40%	

Table 5: Advanced operators per query.

SERP viewed	year 2003 % queries	year 2004 % queries	Δ
1	100.00%	100.00%	0.00%
2	16.76%	14.38%	-2.38%
3	8.56%	7.41%	-1.15%
4	5.20%	4.52%	-0.68%
5	3.57%	3.10%	-0.47%
6	2.54%	2.25%	-0.29%
7	1.93%	1.73%	-0.20%
8	1.51%	1.40%	-0.11%
9	1.25%	1.18%	-0.07%
≥10	5.74%	6.71%	+0.97%

Table 6: SERPs viewed per query.

The small use of advanced operators is in accordance with previous studies [4, 18, 11, 3]. The SITE and PHRASE operators divided the preferences, being used in more than 43% of the advanced queries each. The NOT operator was used in less than 4% of the advanced queries and was insignificantly used when compared to the total number of queries. Overall, it seems that the users were unfamiliar with the advanced operators. Eastman and Jansen suggest that the low presence of advanced operators is due to the little or no benefit they provide [23].

SERPs The users saw on average about 1.4 SERPs per query on the two analyzed years. Table 6 presents the SERPs viewed per query. All users saw the first SERP as expected, since the search engine always returned it after a query. Then, the users followed the natural order of the SERPs, but in a sharp decline. For instance, the second SERP was viewed in 16.76% of the queries. This indicates that prefetching of SERPs would not significantly improve web search engine performance. The results also show slight decreases on all the SERPs viewed. For instance, from 2003 to 2004 the second SERP was viewed less 2.38%. Moreover, the percentage of sessions where the users viewed only the first SERP increased from 68.11% in 2003 to 76.66% in 2004.

Clicks The users clicked around 0.7 times per query to access a web page listed on the SERPs. We analyzed the results they clicked and observed that its distribution fits the power law, with a 0.98 correlation (see Figure 2). This is similar to other studies, which also present a discontinuity in the last ranking position of each SERP (multiple of 10) [24]. The results almost did not vary between the two years. More than 70% of the clicks occurred on the first SERP. This is a good indicator of the ranking quality of the Tumba! web search engine.

Term distribution The distribution of the terms per query listed in Table 7 shows that the length of the queries varied little from 2003 to 2004. The majority of the queries had 1 or 2 terms. This is also visible by the 2.2 average of terms per query (see Table 1). More than 93% of the queries had up to 4 terms and more than 99% up to 7 terms. These results indicate that the users tend to submit short queries, with each term having in average 6.99 characters in 2003 and 6.80 in 2004. These values are useful, for instance, to optimize index structures [25] or to determine the adequate length of the input text boxes on the interface.

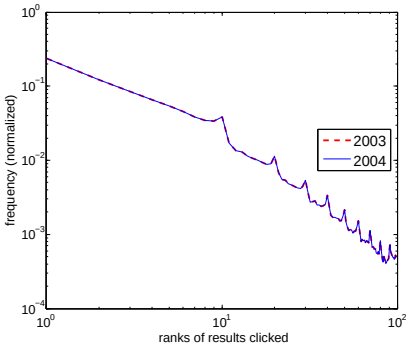


Fig. 2: Distribution of ranks clicked.

# terms	year 2003 % queries	year 2004 % queries	Δ
1	39.30%	39.98%	+0.68%
2	29.00%	26.87%	-2.13%
3	18.66%	18.84%	+0.18%
4	7.04%	7.37%	+0.33%
5	3.27%	3.66%	+0.39%
6	1.41%	1.66%	+0.25%
7	0.68%	0.85%	+0.17%
8	0.29%	0.37%	+0.08%
9	0.15%	0.18%	+0.03%
≥10	0.21%	0.22%	+0.01%

Table 7: Number of terms per query.

Query frequency distribution We ranked the unique queries by their decreasing frequency and verified that its distribution fits the power law as in other studies [9], with a 0.95 correlation. This means that a small number of queries were submitted many times, while a large number of queries were submitted just a few times. The 36,000 and 22,000 most frequent queries in 2003 and 2004, respectively, represent 50% of the total volume of submitted queries. However, they are only 11% in 2003 and 13.56% in 2004, of all the unique queries. Figure 3 depicts the 2003 and 2004 cumulative distributions of queries. We can see that, by caching a little more than 10% of the most frequent queries, the Tumba! search engine could respond to 50% of the query requests.

5.3 Term Level Analysis

Term frequency distribution Analogous to the query frequency distribution, we ranked the unique terms by their decreasing frequency. Its distribution also fits the power law, now with a 0.98 correlation. As depicted in Figure 4, the cumulative distribution shows that it is necessary to cache just around 1% of the most frequent terms, 1,200, to handle 50% of the queries. Much less RAM is necessary to cache terms than queries for a similar hit rate. These results are consistent with the ones presented by Baeza-Yates et al. [9]. However, caching the terms instead of the queries, adds the extra processing over the posting lists to evaluate the results matching the query. A proper tradeoff must be found.

5.4 Topical analysis

Table 8 lists the 20 most frequent queries and terms searched in 2003 and 2004. *Sexo* (sex) is the most searched query in both years, while *emprego* (job) is the second. Then, in 2003, like in 2004, the top 20 contains many queries related with sex and some queries related to the University of Lisbon, such as *mestrados* (master degrees), since Tumba! was also the University's site search engine. Other interests were games, the *totoloto* lottery, maps, chat, postcards, mp3 and music, the Benfica team and the Euro 2004 soccer event, humor and jokes, the eMule

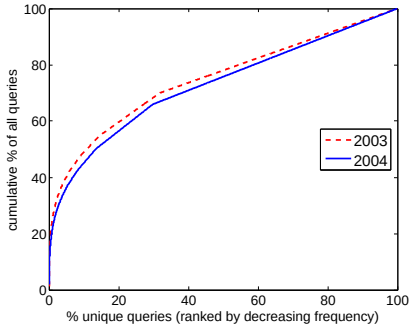


Fig. 3: Cumulative distributions of queries.

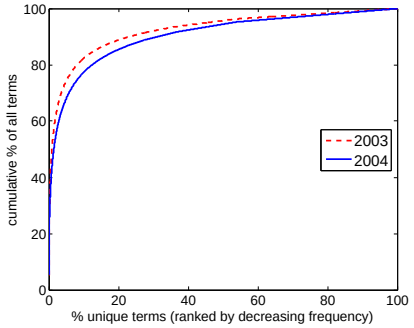


Fig. 4: Cumulative distribution of terms.

P2P program, taxes and photos. As a rough profile, we could say that the main concerns of the Portuguese users are sex, job and entertainment.

With the purpose of determining the types of information that people search for, we manually classified for each year, a random sample of 1,000 queries under the eleven general categories defined by Spink et al [12]. We chose this taxonomy for reasons of comparability with previous studies. To reduce bias, the same queries were independently classified by two evaluators, which then resolved their discrepancies. Table 9 shows the query percentages by topic categories. The most searched category was *Commerce, Travel, Employment or Economy*, with 22.40% of the queries in 2003 and 20.30% in 2004. This broad topic decreased 2.10%. On the other hand, the second most searched category, *People, Places or Things*, increased 2.90%, from 14.80% to 17.70%. Despite the sexual related queries being on the top of the most searched queries, the *Sex or Pornography* topic counts only with 4.90% of the queries in 2003 and 5.80% in 2004. Noteworthy, is also the 3.60% decrease of the *Entertainment or Recreation* topic.

6 Discussion and Conclusion

The Portuguese users, like other users, did not spend much time and effort on individual web searches. The Portuguese users submitted short sessions with short queries and few clicks, did not see beyond the first SERP and rarely used advanced operators. From 2003 to 2004 the average session duration and queries per session decreased. Sessions with less than one minute increased 10% and sessions with only one query increased 9%. The sessions where only the first SERP was viewed increased 8%. These results are in accordance with other results about the European users, which indicate that searching is moving toward a greater simplicity [15]. An analysis over an extended period is necessary to confirm this tendency. However, if verified, web search engines will be receiving less data while they cope with providing the same good results.

The Portuguese users have some peculiarities. The most common modification from users of other studies is to maintain the same number of terms when changing a query [11, 22, 18]. The Portuguese users on the other hand, tend to

rank	year 2003				year 2004			
	query	queries	term	terms	query	queries	term	terms
1	sexo	1.26%	sexo	1.03%	sexo	2.04%	sexo	1.39%
2	emprego	0.29%	portugal	0.40%	emprego	0.24%	portugal	0.42%
3	isep	0.14%	fotos	0.35%	emule	0.22%	fotos	0.33%
4	jogos	0.12%	lisboa	0.32%	jogos	0.15%	lisboa	0.28%
5	totoloto	0.10%	emprego	0.26%	chat	0.13%	jogos	0.24%
6	escola	0.10%	escola	0.23%	pornografia	0.13%	imagens	0.22%
7	mestrados	0.10%	porto	0.22%	totoloto	0.13%	2004	0.21%
8	pornografia	0.09%	jogos	0.21%	f******	0.12%	emprego	0.20%
9	porno	0.09%	imagens	0.17%	porno	0.10%	emule	0.20%
10	mapas	0.08%	mapa	0.17%	cadastro comercial	0.10%	download	0.19%
11	cadi	0.08%	trabalho	0.17%	anedotas	0.10%	porto	0.18%
12	chat	0.08%	gratis	0.16%	irs	0.09%	escola	0.18%
13	postais	0.07%	download	0.16%	travestis	0.08%	mapa	0.17%
14	mp3	0.07%	cursos	0.15%	mestrados	0.08%	lei	0.17%
15	benfica	0.07%	portuguesa	0.15%	euro 2004	0.08%	escolas	0.16%
16	humor	0.07%	universidade	0.15%	tumba	0.07%	gratis	0.16%
17	contos eroticos	0.07%	formação	0.14%	google	0.07%	trabalho	0.14%
18	acompanhantes	0.07%	2003	0.14%	contos eroticos	0.07%	comercial	0.14%
19	anedotas	0.07%	musica	0.14%	horarios	0.07%	portuguesa	0.14%
20	sexo gratis	0.07%	ensino	0.14%	sexo gratis	0.07%	cursos	0.14%

Table 8: The 20 most frequent searched queries and terms. Characters **** hide expletives.

	Categories	year 2003	year 2004	Δ
		% queries	% queries	
1	Commerce, Travel, Employment or Economy	22.40%	20.30%	-2.10%
2	People, Places or Things	14.80%	17.70%	2.90%
3	Health or Sciences	10.50%	11.80%	1.30%
4	Education or Humanities	7.20%	10.50%	3.30%
5	Society, Culture, Ethnicity or Religion	5.60%	6.10%	0.50%
6	Computers or Internet	6.40%	5.90%	-0.50%
7	Sex or Pornography	4.90%	5.80%	0.90%
8	Entertainment or Recreation	8.70%	5.10%	-3.60%
9	Government	7.00%	4.20%	-2.80%
10	Performing or Fine arts	1.60%	1.60%	0.00%
11	Unknown or Other	11.20%	11.30%	0.10%

Table 9: Topic categories of the queries.

refine the query by adding a term. For the other searching aspects, we can make the rough generalization that the European users submit less information to satisfy an information need than the U.S. users, but compensate by seeing more SERPs [6, 1]. The Portuguese users, which are also European, submit even less information and see even less SERPs than the other users. Table 10 summarizes these findings. We can speculate that these differences are due to the cultural differences of users, which are less tolerant and give up more easily. Other hypothesis is that the differences are due to the superior results' quality or superior interface that enabled the users to find the information sooner. The analysis over the search logs is insufficient to respond this question.

An important finding of this study is that the specificities of the Portuguese users do not preclude the general adoption of searching technology used by the U.S. and European users. On the other hand, the identification of these specificities can contribute to the development of better adapted web search engines. For instance, our results show that caching around 1% of the most frequent query terms enables response to 50% of the queries and caching the last query of a user in a session enables response to near a third of the queries.

world region search engine	U.S. Excite	Europe FAST	Portugal Tumba!
single term queries	20% - 30%	25% - 35%	40%
terms per query	2.6	2.3	2.2
queries per session	2.3	2.9	2.49 - 2.94
advanced queries	11% - 20%	2% - 10%	11% - 13%
SERPs viewed	1.7	2.2	1.4
topic most seen	Commerce, Travel	People, Places	Commerce, Travel

Table 10: General comparisons between users.

References

1. Jansen, B., Spink, A.: How are we searching the World Wide Web? A comparison of nine search engine transaction logs. Information Processing and Management **42**(1) (2006) 248–263
2. Markey, K.: Twenty-five years of end-user searching, Part 1: Research findings. American Society for Information Science and Technology **58**(8) (2007) 1071–1081
3. Aula, A., Khan, R.M., Guan, Z.: How does search behavior change as search becomes more difficult? In: Proc. of the 28th International Conference on Human Factors in Computing Systems. (2010) 35–44
4. Hölscher, C., Strube, G.: Web search behavior of Internet experts and newbies. Computer networks **33**(1-6) (2000) 337–346
5. Park, S., Ho Lee, J., Jin Bae, H.: End user searching: A Web log analysis of NAVER, a Korean Web search engine. Library and Information Science Research **27**(2) (2005) 203–221
6. Spink, A., Ozmutlu, S., Ozmutlu, H.C., Jansen, B.J.: U.S. versus European Web searching trends. SIGIR Forum **36**(2) (2002) 32–38
7. Costa, M.: Sidra: a flexible web search system. Master's thesis, University of Lisbon, Faculty of Sciences (November 2004) Also available as Technical Report DI/FCUL TR 4-17.
8. Barbaro, M., Zeller, T.: A face is exposed for AOL searcher No. 4417749. New York Times **9** (2006) `http://www.nytimes.com/2006/08/09/technology/09aol.html`.
9. Baeza-Yates, R., Gionis, A., Junqueira, F.P., Murdock, V., Plachouras, V., Silvestri, F.: Design trade-offs for search engine caching. ACM Transactions on the Web **2**(4) (2008) 1–28
10. Hearst, M.: Search User Interfaces. Cambridge University Press (2009)
11. Jansen, B.J., Spink, A., Saracevic, T.: Real life, real users, and real needs: a study and analysis of user queries on the Web. Information Processing and Management **36**(2) (2000) 207–227
12. Spink, A., Jansen, B., Wolfram, D., Saracevic, T.: From e-sex to e-commerce: Web search changes. IEEE Computer **35**(3) (2002) 107–109
13. Lawrence, S., Giles, C.: Accessibility of information on the web. Intelligence **11**(1) (2000) 32–39
14. Jansen, B., Spink, A., Pedersen, J.: A temporal comparison of AltaVista Web searching. American Society for Information Science and Technology **56**(6) (2005) 559–570
15. Jansen, B., Spink, A.: An analysis of Web searching by European AlltheWeb.com users. Information Processing and Management **41**(2) (2005) 361–381
16. Beitzel, S., Jensen, E., Chowdhury, A., Frieder, O., Grossman, D.: Temporal analysis of a very large topically categorized web query log. American Society for Information Science and Technology **58**(2) (2007) 166–178
17. Ozmutlu, S., Ozmutlu, H., Spink, A.: Multitasking Web searching and implications for design. American Society for Information Science and Technology **40**(1) (2003) 416–421
18. Silverstein, C., Marais, H., Henzinger, M., Moricz, M.: Analysis of a very large web search engine query log. In: ACM SIGIR Forum. Volume 33. (1999) 6–12
19. Cacheda, F., Vina, A.: Understanding how people use search engines: a statistical analysis for e-business. E-work and e-commerce: novel solutions and practices for a global networked economy (2001) 319
20. Jones, R., Klinkner, K.L.: Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In: Proc. of the 17th ACM Conference on Information and Knowledge Management, New York, NY, USA, ACM (2008) 699–708
21. Radlinski, F., Joachims, T.: Query chains: learning to rank from implicit feedback. In: Proc. of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. (2005) 239–248
22. Spink, A., Wolfram, D., Jansen, M., Saracevic, T.: Searching the web: The public and their queries. American Society for Information Science and Technology **52**(3) (2001) 226–234
23. Eastman, C., Jansen, B.: Coverage, relevance, and ranking: The impact of query operators on Web search engine results. ACM Transactions on Information Systems (TOIS) **21**(4) (2003) 383–411
24. Baeza-Yates, R., Hurtado, C., Mendoza, M., Dupret, G.: Modeling user search behavior. In: Proc. of the 3rd Latin American Web Congress. (2005) 242
25. Lucchese, C., Orlando, S., Perego, R., Silvestri, F.: Mining query logs to optimize index partitioning in parallel web search engines. In: Proc. of the 2nd International Conference on Scalable Information Systems. (2007) 1–9

Extracção de conhecimento léxico-semântico a partir de resumos da Wikipédia

Hugo Gonçalo Oliveira*, Hernani Costa, Paulo Gomes
hroliv@dei.uc.pt, hpcosta@student.dei.uc.pt, pgomes@dei.uc.pt

Cognitive and Media Systems Group
Centro de Informática e Sistemas
Universidade de Coimbra, Portugal

Resumo Este artigo apresenta um sistema para a aquisição automática de relações semânticas a partir de texto em português, o que pode ser visto como um passo central na construção automática de recursos léxico-semânticos. O sistema foi aplicado à Wikipédia, actualmente uma enorme fonte de conhecimento livre. Os resultados obtidos e a sua avaliação são discutidos, as actuais limitações referidas e são ainda apresentadas várias ideias para futuras melhorias.

Abstract This paper presents a system for the automatic acquisition of semantic relations from Portuguese text, which can be seen as core step in the automatic construction of lexico-semantic resources. The system was applied to Wikipedia, currently a huge and free source of knowledge. The obtained results are shown and their evaluation is discussed together with the current limitations and cues for further improvement.

1 Introdução

A realização de tarefas, cada vez mais comuns, onde é necessário compreender as interacções entre as palavras e os seus significados, tal como a resposta automática a perguntas, a tradução automática ou a recuperação de informação, levou à criação de recursos semânticos computacionais de larga cobertura, como as ontologias lexicais, de onde se destaca, para o inglês, a WordNet de Princeton [9]. No entanto, a construção e a manutenção deste tipo de recurso envolve muito trabalho intensivo, realizado por humanos. De forma a contornar este problema, têm nas últimas décadas surgido várias propostas para, a partir de texto, extrair automaticamente conhecimento léxico-semântico que pode ser utilizado para criar ou para ampliar uma ontologia lexical.

Estas abordagens têm sido aplicadas a diferentes tipos de texto, e conhecimento léxico-semântico vem sendo extraído a partir de recursos estruturados, como os dicionários [6] [17] [12], ou não estruturados, como os corpos [13] [5] [10]. Se por um lado há vantagens em utilizar dicionários, por estes se encontrarem já estruturados em palavras e significados e ainda

* Financiado pela bolsa FCT SFRH/BD/44955/2008

por utilizarem um vocabulário simples, quase previsível, este tipo de recurso contém conhecimento limitado, é normalmente estático e nem sempre se encontra disponível para fins de investigação. Por outro lado, hoje em dia é possível encontrar muito texto pela Web, praticamente acerca de qualquer assunto, mas cujo processamento não é tão simples devido à existência de menos restrições sintácticas e à utilização de vocabulário mais variado e mais ambíguo. Um terceiro tipo de recurso, que podemos considerar semi-estruturado, é a enciclopédia, onde existem também entradas para diferentes entidades, mas cujas descrições são mais extensas, podendo ser encaradas como texto de corpos. Além disso, o conteúdo das enciclopédias não se limita a informação sobre as palavras e inclui mais conhecimento sobre o mundo e saber humano.

Assim, também devido à sua disponibilidade na Web, no últimos anos tornou-se frequente a utilização de enciclopédias, como a Wikipédia[1], para extrair informação. Tendo em conta a sua construção colaborativa, este recurso é uma enorme fonte de informação em permanente evolução. Para o inglês, a Wikipédia foi já utilizada numa grande quantidade de tarefas, onde destacamos a extracção de relações taxonómicas [14] e de outras relações léxico-semânticas, com vista ao enriquecimento da WordNet [19]. A utilidade da Wikipédia na extracção de conhecimento léxico-semântico é apontada por [21], que implementaram um interface para o acesso programático a este recurso e também ao Wikcionário. Além disso, a descrição de alguns trabalhos que utilizam a Wikipédia para extrair conceitos, relações, factos e descrições pode encontrar-se em [15]. Também para o português a Wikipédia se revelou ser um importante recurso, por exemplo no apoio à identificação de entidades mencionadas (EM) [4].

O trabalho aqui descrito enquadra-se num projecto que tem como objectivo final a construção automática de uma ontologia lexical para o português onde, entre outros recursos, a Wikipédia é também explorada. Mais precisamente, são extraídas relações semânticas a partir dos resumos da versão portuguesa da Wikipédia de forma a obter informação que pode ser utilizada para criar um novo recurso ou para enriquecer recursos lexicais já existentes, como o PAPEL [11], uma rede lexical extraída automaticamente a partir de um dicionário.

Começamos por apresentar as fases do nosso sistema que se baseia num conjunto de gramáticas semânticas, onde estão presentes padrões textuais indicadores de relações. De seguida descrevemos a experimentação realizada que inclui: a extracção de triplos a partir da Wikipédia; a análise dos resultados; a avaliação manual de uma amostra de resultados; a análise dos principais padrões textuais que originaram triplos; e uma proposta para avaliação automática, cuja utilidade não foi contudo comprovada. Por fim concluímos ao apontar algumas limitações actuais do sistema e referimos ideias para trabalho futuro.

2 Extracção automática de relações semânticas

O sistema de extracção de relações semânticas que estamos a desenvolver é constituído por vários módulos (ver figura 1) e está centrado num conjunto de

[1] http://wikipedia.org

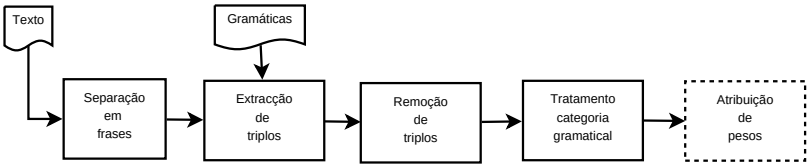


Figura 1. Os módulos do sistema de extracção.

gramáticas semânticas, construídas com base em padrões que indicam relações em texto escrito em português. Até ao momento, o sistema extrai relações de sinonímia, hiperonímia, parte, causa e finalidade. Exemplos destas relações e de alguns dos padrões ou palavras chave utilizados na sua extracção podem encontrar-se na apresentação dos resultados obtidos, mais propriamente na tabela 1 e na tabela 3. Como o sistema está preparado para analisar texto frase a frase, o primeiro módulo prepara o texto fornecido, separando-o em frases. Na fase de extracção, cada frase é analisada e é obtida uma uma árvore de derivação por gramática. Em cada árvore o sistema procura por nós que identificam um padrão, dentro dos quais poderão existir nós identificadores dos argumentos de uma relação, cujo conteúdo serão termos, ou enumerações de termos, a ser combinados num triplo relacional. Por exemplo, ao encontrar os nós HIPERONIMO e HIPONIMO, o sistema vai extrair o triplo *hiper* HIPERONIMO_DE *hipo*, em que *hiper* e *hipo* são respectivamente os conteúdos de HIPERONIMO e HIPONIMO.

Na versão actual das gramáticas optámos por extrair relações entre termos compostos, ou seja, se um termo ocorrer modificado por um adjectivo (p.e. *computador pessoal*) ou por uma preposição (p.e. *sistema de controlo*) é extraído dessa forma, e pode dar origem a um termo com várias palavras (p.e. *movimento de massa exclusivo das regiões vulcânicas*). Futuramente, após avaliar a relevância destes termos, será possível tomar decisões relativamente à sua manutenção, possibilitando também uma melhor organização do recurso.

Ainda na fase de extracção, o sistema tira partido de dois padrões léxico-sintácticos, `[N ADJ]` e `[N de|do|da|com|para N]`, para obter relações de hiperonímia a partir de termos compostos. Por exemplo, a partir dos termos *computador pessoal* e *sistema de controlo* são extraídos respectivamente os triplos *computador* HIPERONIMO_DE *computador_pessoal* e *sistema* HIPERONIMO_DE *sistema_de_controlo*. Neste tipo de extracção, o segundo padrão mencionado não é aplicado se o primeiro `N` se tratar de uma palavra sem conteúdo (p.e. *tipo*, *forma*) ou que implique uma relação de parte (p.e. *parte*, *membro*, *grupo*, *conjunto*) e não de hiperonímia, chamadas, no contexto da análise de dicionários, cabeças vazias (do inglês *empty heads*)[6].

Para identificar as categorias gramaticais das palavras é previamente realizada a análise morfo-sintáctica de cada frase, utilizando um modelo para o *pos-tagger* fornecido no pacote OpenNLP[2], treinado com o Bosque, uma

[2] http://opennlp.sourceforge.net/

parte do treebank Floresta Sintá(c)tica [1] completamente revista por linguistas. No entanto, as gramáticas contêm essencialmente padrões lexicais e apoiam-se nas categorias gramaticais apenas para identificar adjectivos. Além da análise morfo-sintáctica, cada palavra da frase é lematizada, também recorrendo a um modelo do OpenNLP a que foi acrescentado um pequeno conjunto de regras para passagem de plural para singular.

Após a extracção, triplos cujos argumentos estejam numa lista de palavras não pretendidas (essencialmente *stopwords*) são removidos. A penúltima fase possibilita remover triplos ou alterar o nome da sua relação com base na categoria gramatical dos seus argumentos, baseando-se numa especificação onde, para cada nome de relação extraída, poderá existir um segundo nome de acordo com a categoria gramatical dos seus argumentos. Se pretendido, é também possível lematizar os argumentos dos triplos, com base no lema obtido anteriormente.

Estamos ainda a ponderar a inclusão de uma fase em que os triplos recebem pesos de acordo não só com a frequência com que foram extraídos, mas também com o valor de métricas distribucionais calculadas na Web ou numa colecção de documentos, tal como [5] ou [20] sugerem. A este respeito verificamos [7] que a qualidade de triplos de hiperonímia e também de parte está correlacionada com o valor de algumas métricas distibucionais em corpos, como o LSA e o coeficiente de Jaccard. Os pesos poderão depois ser utilizados para eliminar triplos pouco relevantes ou cuja probabilidade de estarem correctos seja baixa.

3 Experimentação com a Wikipédia

Para testar o nosso sistema optámos por aplicá-lo a resumos da versão portuguesa da Wikipédia. Estes foram escolhidos por descreverem em poucas palavras o conteúdo de cada artigo, tendo por isso a informação mais relevante nele contida e menos variações ao nível da estrutura.

3.1 Preparação

Cedo verificámos que grande parte dos conteúdos da Wikipédia são demasiado específicos para serem centrais na construção de uma ontologia lexical, como é o caso de artigos sobre personalidades, organizações ou épocas históricas. Devido a este problema procuramos uma forma de filtrar resumos associados a EM, o que levaria também a uma diminuição da quantidade de texto a processar.

Para tal, utilizamos os resumos disponibilizados pelo projecto DBpedia [2] e a taxonomia definida no seu âmbito. Com vista à construção de uma base de conhecimento, a DBpedia mapeia a Wikipédia numa taxonomia onde a cada artigo é atribuído um ou vários tipos de alto nível, como por exemplo *Person*, *Place*, *Organization*, *MeanOfTransportation*, *Device* ou *Species*, além de tipos mais específicos como *Writer*, *Airport*, *SoccerClub*, *Bird*, *Automobile* ou *Weapon*.

Ainda que a atribuição de tipos não se encontre disponível para a versão portuguesa da Wikipédia, há uma correspondência entre os identificadores das entradas nas várias línguas que dizem respeito ao mesmo assunto. Por isso

utilizamos os tipos atribuídos às entradas da versão inglesa para filtrar da versão portuguesa entradas do tipo *Person*, *Place*, *Organization*, *Event*, entre outros associados a EM. Apesar de haver várias entradas da Wikipédia portuguesa sem correspondência, cerca de 30% dos 368.521 resumos originais foi removido, perfazendo as 494.187 frases a que chamaremos o conjunto de resumos A.

Ainda assim, ficamos com muito texto que não nos interessava processar, de onde destacamos entradas acerca de geografia portuguesa e brasileira. Por isso, à custa de perdermos entradas interessantes e que só existem na Wikipédia portuguesa, optamos por diminuir ainda mais o conjunto de frases mantendo apenas entradas que, através da taxonomia, conseguimos confirmar pertencerem aos tipos: *Species*, *AnatomicalStructure*, *ChemicalCompound*, *Disease*, *Currency*, *Drug*, *Activity*, *Language*, *MusicGenre*, *Colour*, *EthnicGroup* e *Protein*. Além disso, apesar de vários resumos serem constituídos por duas ou três frases, optámos por utilizar apenas a primeira frase de cada um. Desta forma ficamos com um total de 37.898 frases para processar, que constituem o conjunto B, aquele que mais exploramos nesta experiência.

3.2 Resultados

As quantidades de triplos extraídos a partir de ambos os conjuntos (A e B), antes (Total) e depois (S/rep) de remover triplos repetidos, são apresentadas na tabela 1 juntamente com alguns exemplos. Para a hiperonímia separamos os triplos extraídos a partir da análise de termos compostos (TC) dos triplos extraídos através da identificação de padrões textuais.

Relação	**Extraídos A**		**Extraídos B**		**Exemplos**
	Total	S/rep	Total	S/rep	
Hiperonímia TC	711.954	390.492	24.367	16.228	(*desordem*,*desordem_cerebral*) (*átomo*,*átomo_de_carbono*)
Hiperonímia	149.845	144.839	31.254	29.563	(*desporto*,*automobilismo*) (*estilo_de_música*,*folk*)
Sinonímia	25.816	25.518	11.872	11.862	(*inglês_antigo*,*anglo-saxão*) (*estupro*,*violação*)
Parte	12.093	11.485	1.321	1.287	(*jejuno*,*instestino*) (*rolas*,*columbidae*)
Finalidade	13.277	12.992	777	743	(*amoxicilina*,*tratamento_de_infecções*) (*construção*, *terracota*)
Causador	5.854	5.740	559	520	(*parasita*, *doença*) (*doença_neuromuscular*,*fadiga*)

Tabela 1. Resultados totais da extracção em triplos.

Verifica-se que, de ambos os conjuntos, foi extraído um grande número de relações de hiperonímia através da análise de padrões textuais. Isto explica-se porque muitos resumos começam com a construção `[X é um Y]`, resultando em X HIPERONIMO_DE Y. Além disso, há frases com

uma enumeração no lugar de X, o que dá imediatamente origem a uma relação de hiperonímia por cada termo enumerado. Por exemplo, a frase *A heroína ou diacetilmorfina é uma droga* dá origem a: *droga* HIPERONIMO_DE *heroína*, *droga* HIPERONIMO_DE *diacetilmorfina*, *heroína* SINONIMO_DE *diacetilmorfina* e *diacetilmorfina* SINONIMO_DE *heroína*.

Outras curiosidades estão relacionadas com o âmbito dos triplos extraídos. As relações de hiperonímia atribuem essencialmente um género, espécie ou ordem a plantas, animais ou outros seres vivos. As relações de finalidade associam normalmente problemas de saúde às suas terapêuticas, e as relações de causa também se estabelecem muitas vezes entre problemas de saúde, suas causas e efeitos. Já as relações de sinonímia são por vezes estabelecidas entre termos na variante europeia e na variante brasileira do português, como por exemplo em *marrom* SINONIMO_DE *castanho* ou *esófago* SINONIMO_DE *esôfago*. Além disso, muitas das frases de onde são extraídas relações de sinonímia são iniciadas pela enumeração de uma grande quantidade de sinónimos. O caso extremo desta situação é a frase iniciada por: *Bagre-bandeira, bagre-cacumo, bagre-de-penacho, bagre-do-mar, bagre-fita, bagre-mandim, bagre-sari, bandeira, bandeirado, bandim, pirá-bandeira, sarassará, sargento ou bagre-bandeirado ... é um peixe da família dos ariídeos....*

3.3 Avaliação manual

A primeira abordagem à avaliação dos nossos resultados foi feita manualmente, através da classificação de um grupo de triplos seleccionado aleatoriamente de acordo com a escala proposta em [10], que sugere a classificação de triplos em quatro grupos: correctos (3); com uma preposição ou um adjectivo que deixam um dos argumentos estranho e impede o triplo de estar correcto (2); correcto, mas demasiado geral ou específico para ter utilidade (1); incorrecto (0).

Assim, foram inicialmente geradas 12 amostras aleatórias com 85 triplos extraídos a partir do conjunto A, classificadas cada uma por dois revisores. Para confirmarem a qualidade dos triplos, os revisores foram aconselhados a procurar na Web, incluindo a própria Wikipédia, por informação acerca das entidades envolvidas. A utilização desta escala permitiu por um lado identificar triplos que, devido a algum problema com as regras das gramáticas, deu origem a argumentos incompletos, e por outro identificar triplos que apesar de estarem correctos, não têm grande utilidade prática, principalmente no âmbito de uma ontologia lexical. Nesta categoria, encontram-se triplos que indicam subdivisões geográficas (p.e. *sub-região_estatística_portuguesa* PARTE_DE *região_do_alentejo*), relacionados com épocas históricas (p.e. *tragédia_de_1892* CAUSADOR_DE *crise_política*), entre outros demasiado específicos (p.e. *romancista_brasileiro* PARTE_DE *academia_brasileira_de_letras*, *escola* HIPERONIMO_DE *escola_de_música_Juilliard*).

Além disso, utilizamos a especificação de relações utilizada no PAPEL [12] para tratar o nome de cada triplo de acordo com as categorias gramaticais dos seus argumentos. No entanto verificamos que, essencialmente devido a limitações

do *pos-tagger*, mas também devido ao género de texto processado, a grande maioria dos triplos cujo nome era alterado devido à categoria de um, ou ambos, os argumentos não ser substantivo, estava incorrecto. Optamos então por prosseguir a avaliação utilizando apenas relações cujos argumentos eram identificados como substantivos.

Tendo isto em conta, foram gerados novos dados para teste com triplos do conjunto B. Para tal, utilizamos 663 triplos que já tinham sido classificados na primeira avaliação e se mantinham no conjunto B, aos quais juntamos mais 12 amostras aleatórias, com cerca de 90 triplos cada uma, avaliadas da mesma forma que as primeiras.

Os resultados da segunda avaliação encontram-se na tabela 2, onde as proporções apresentadas somam as avaliações dos dois revisores, a que juntamos a concordância exacta entre ambos (CcEx) e a concordância relaxada (CcRel), em que os valores 1 e 3 foram considerados correctos e 0 e 2 incorrectos, atendendo a que estes resultados poderiam vir a ser utilizada noutro âmbito e os triplos classificados com 1 também estão correctos.

Um dado saliente ao comparar os resultados obtidos com o conjunto A com os obtidos com o conjunto B é a diferença do número de triplos classificados com 1. Em termos de proporção, este número decresceu de 39% para 22% do total de triplos. Também em proporção, houve um aumento de triplos correctos. Por exemplo, os triplos classificados com 3 aumentaram aproximadamente 2 e 1,5 vezes nas relações de finalidade e causa. As melhorias dever-se-ão ao conjunto B ser mais restrito, com uma construção mais próxima e onde existirá menor ambiguidade. Ainda assim, cerca de um quarto dos triplos de causa e finalidade e um quinto dos triplos de parte continua completamente errado, o que estará essencialmente relacionado com a ambiguidade de alguns padrões utilizados.

Na tabela 2 verifica-se ainda uma maior concordância na divisão entre triplos correctos e incorrectos, essencialmente por se tratar de uma divisão mais objectiva, onde não entra a subjectividade de avaliar a utilidade efectiva de um triplo numa ontologia lexical. Por exemplo, vários triplos de hiperonímia extraídos através de termos compostos não acrescentam muito à base de conhecimento (p.e. *equipa* HIPERONIMO_DE *equipa_de_seis_jogadores*), mas esta classificação é bastante sensível ao critério do revisor.

Há no entanto um ponto em que esta avaliação piorou, mais propriamente na proporção de triplos de hiperonímia classificados com 2. Isto acontece porque a proporção de frases sobre espécies aumentou e muitas destas espécies são identificadas por duas palavras. Por exemplo, na frase *O Iriatherina werneri é uma espécie de peixe de aquário*, o *pos-tagger* não conhece as duas palavras da entidade *Iriatherina werneri*, o que leva o sistema a não interpretar a entidade como um substantivo modificado e, por isso, a extrair um triplo com um argumento incompleto, *peixe_de_aquário* HIPERONIMO_DE *werneri*.

3.4 Eficiência dos padrões

Além de avaliar a qualidade dos triplos extraídos, também nos pareceu interessante fazer o levantamento dos padrões ou palavras chave que davam

Relação	Avaliados	3(%)	2(%)	1(%)	0(%)	CcEx(%)	CcRel(%)
Hiperonímia TC	323	35,0	4,2	42,1	18,7	57,3	82,7
Hiperonímia	322	57,5	33,8	1,6	7,1	89,8	93,1
Sinonímia	286	85,7	7,3	0,4	6,6	90,0	91,6
Parte	268	44,2	26,7	8,4	20,7	63,1	78,4
Finalidade	264	53,0	16,5	4,0	26,5	71,2	82,2
Causador	244	41,8	24,6	7,8	25,8	61,5	79,5

Tabela 2. Resultados da avaliação manual de triplos.

origem a mais triplos. A esses dados, que para o conjunto B se encontram na tabela 3, juntamos informação acerca da classificação obtida na avaliação manual por triplos extraídos através destes padrões. Neste caso, apenas considerámos triplos onde a avaliação de ambos os revisores era concordante. Dentro dos padrões que levam à extracção de mais triplos incorrectos, destacamos `[usado|utilizado]` que, quando seguido de `[em|no|na]` pode não indicar a relação de finalidade, mas sim um local onde um objecto é utilizado, como em *O Ariary malgaxe é a moeda usada em Madagáscar*. Outro padrão bastante ambíguo parece ser `[inclui|incluem]`. Por outro lado, a utilização do padrão `é um género de` apenas levou à extracção de triplos de hiperonímia correctos.

Relação	Padrão	Extraídos	Avaliados			
			3	2	1	0
Hiperonímia	*termo composto*	24.367	72	7	75	32
Hiperonímia	`é uma espécie de`	15.824	54	96	0	0
Hiperonímia	`é um|uma`	10.960	87	11	0	15
Hiperonímia	`é um género de`	2.402	24	0	0	0
Sinonímia	`ou`	4.886	154	2	0	2
Sinonímia	`também conhecido|a|os|as por|como`	3.016	60	4	0	4
Parte	`inclui|incluem`	471	34	0	2	15
Parte	`grupo de`	158	17	3	1	0
Finalidade	`utilizado|a|os|as para|como|em|no|na`	376	71	16	1	20
Finalidade	`usado|a|os|as para|como|em|no|na`	237	41	3	1	4
Causador	`causado|a|os|as`	165	27	11	1	10

Tabela 3. Triplos extraídos e sua qualidade de acordo com o padrão utilizado.

3.5 Proposta para validação automática

Como é sabido, ainda que seja provavelmente a forma mais confiável de avaliação, a avaliação manual de relações semânticas é um trabalho moroso e cansativo, além de ser muitas vezes subjectivo por mais critérios que sejam definidos. Isto confirma-se pelas taxas de concordância que obtivemos na nossa avaliação manual. Ainda que tenhamos utilizado duas formas para medir a concordância,

nem sempre é fácil distinguir entre as várias classificações de uma escala. Por exemplo, além da subjectividade existente ao decidir a utilidade de um triplo, a distinção entre a classificação 1 e 2 pode não ser muito clara, já que o triplo pode ser muito geral, ou específico, exactamente por lhe faltar um modificador. Além disso, este tipo de avaliação não é facilmente repetível, o que não se passaria se existisse um método automático para avaliar a qualidade dos resultados. Com isto em mente, surgiu a nossa primeira abordagem a uma avaliação automática.

Uma das formas que vem sendo comum para validar, de forma automática, dados resultantes da extracção de informação passa por tirar partido da enorme quantidade de informação disponível na Web. No caso específico da validação de triplos semânticos, uma alternativa seria procurar por frases em que a relação entre ambos os argumentos está explícita através de padrões textuais. Isto é feito por exemplo em [12], mas sobre um corpo de notícias.

Seguindo estas ideias, a validação automática dos triplos extraídos no âmbito deste trabalho teria por base a aplicação de quatro métricas vulgarmente utilizadas para avaliar, na Web, a semelhança entre dois termos [3], mais precisamente: WebJaccard (1), WebOverlap (2), WebPMI (4) e WebDice (3). Nestas equações, $P(X)$ refere-se ao número de páginas em que o termo X ocorre e $P(X \cap Y)$ é o número de páginas em que X e Y co-ocorrem. Na equação 4, N deveria ser o total de páginas indexadas no motor de pesquisa que, não sendo calculável, poderá ser aproximado a 10^{10} [3].

$$WebJaccard(X,Y) = \frac{P(X \cap Y)}{P(X) + P(Y) - P(X,Y)} \tag{1}$$

$$WebOverlap(X,Y) = \frac{P(X \cap Y)}{\min\left(P(X), P(Y)\right)} \tag{2}$$

$$WebDice(X,Y) = \frac{2 * P(X \cap Y)}{P(X) + P(Y)} \tag{3}$$

$$WebPMI(X,Y) = \log_2 \left(\frac{P(X \cap Y)}{P(X) * P(Y)} * N \right) \tag{4}$$

As medidas acima referidas são normalmente utilizadas no cálculo da semelhança distribucional entre dois termos, ou seja, a semelhança dos termos com base nas suas ocorrências e vizinhanças, e, ainda que termos relacionados tenham habitualmente distribuições semelhantes, estas métricas não têm nenhuma relação semântica específica em vista. Sendo assim, inspirados por [16], para aplicarmos estas métricas à validação de triplos semânticos, deverá ser incluído também um padrão textual frequente indicador da relação, ou seja $X = XR$, $Y = RY$ e $X \cap Y = XRY$, sendo R o padrão. A tabela 4 contém padrões que podem ser utilizados para cada relação, depois de observar aqueles que mais frequentemente extraíram triplos (tabela 3). Curiosamente os padrões que extraem mais triplos indicam a relação inversa, ou seja, por exemplo, para validar o triplo t_1 $RELACAO$ t_2, $X = t_2$ e $Y = t_1$.

O primeiro passo foi calcular estas métricas para cada triplo avaliado manualmente em que a classificação fosse concordante para ambos os revisores. Para cada triplo e padrão relativo à sua relação (ver versão simplificada na

tabela 4), calculamos as métricas com base no Google. Logo aí verificamos que obtínhamos valores apenas para uma pequena quantidade de triplos (20% dos concordantes), porque os restantes nunca co-ocorriam com o padrão escolhido. Isto é compreensível, tendo em conta que termos semanticamente relacionados podem co-ocorrer de várias formas ou, por outras palavras, cada relação semântica pode ser traduzida numa enorme quantidade de padrões textuais. Outras limitações estão relacionadas com a própria pesquisa do Google, que não é suficientemente versátil para englobar um grande número de expressões. Além disso, ao procurar por um termo flexionado, o Google não consegue procurar por termos com o mesmo lema, o que limita as pesquisas deste tipo.

Ainda assim, passamos ao passo seguinte onde pretendíamos verificar se existia uma correlação entre os valores obtidos com as métricas para cada tipo de relação e a avaliação humana. Contudo, devido aos factores já referidos, a que acrescentamos a pouca quantidade de triplos disponíveis para esse cálculo, obtivemos sempre valores de correlação baixa, que nunca ultrapassavam os 20%, mesmo transformando a escala da avaliação manual numa escala apenas com 0s e 1s (semelhante à considerada para o cálculo da concordância relaxada).

No futuro pretendemos continuar a nossa busca por um método de validação automática para este trabalho e queremos ainda experimentar estas métricas em corpos para os quais exista um interface de pesquisa mais versátil, como o serviço AC/DC [8].

Relação	**Padrão indicador (R)**
Hiperonímia	é|são um|uma
Sinonímia	também conhecido|conhecida|chamado|chamada|designado|designada de|por|pela
Parte-de	tem|possui|engloba|abrange|inclui|têm um|uma|vários|alguns|
Causa	devido|derivado|derivada|causado|causada|resultado|efeito|consequência a|ao|à|por|pelo|pela| de|do|da
Finalidade	usado|usada|utilizado|utilizada|através|objectivo|finalidade|intuito|serve no|na|para|de o|a|um|uma

Tabela 4. Triplos extraídos e sua qualidade de acordo com o padrão utilizado.

4 Discussão e trabalho futuro

Apresentamos neste artigo o nosso sistema de extracção de relações semânticas a partir de texto não estruturado escrito em português e a sua aplicação a resumos da Wikipédia. O conhecimento extraído, já estruturado, pode ser de grande utilidade no aumento de recursos lexicais para a nossa língua. Nesse contexto, seria interessante realizar uma análise à quantidade de conhecimento extraído que ainda não se encontra no recurso em causa, uma pouco à imagem do que Hearst [13] fez para a WordNet.

Como se pode observar pelos resultados da avaliação, há ainda um longo caminho a percorrer e o sistema tem várias limitações, não só relacionadas com a ambiguidade e com a enorme possibilidade de formas para indicar uma

relação semântica, mas também relacionadas com o *pos-tagger* utilizado e o lematizador, que quando não reconhecem uma palavra procuram inferir a sua categoria gramatical com base em probabilidades e o seu lema com base em regras. Torna-se por isso, para já, impossível obter triplos cujos argumentos estejam lematizados, pois correríamos o risco de deteriorar a sua qualidade. Procuraremos ultrapassar esta limitação com a utilização de outro *pos-tagger* ou analisador morfológico.

Apesar de termos encontrado uma forma de filtrar quase todas as EM, através da taxonomia da DBpedia, haverá ainda várias entradas relevantes para o nosso recurso que ocorrem apenas na Wikipédia portuguesa e estão, desta forma, a ser filtradas sem necessidade. Por isso continuaremos em busca de uma filtragem mais adequada às nossas necessidades, e que poderá tirar partido de outra informação disponível na Wikipédia.

Além de questões já referidas ao longo da descrição da experimentação, e de experiências com métricas de semelhança distribucional, algo que também queremos realizar no futuro é definir um método para aferir a relevância de relações de hiperonímia obtidas através da análise de termos compostos. Por um lado, há uma pequena parte de triplos que podem ser obtidos desta forma e que não estão correctos (p.e. *bola de berlim* não é uma *bola* e *pé de atleta* não é um *pé*) e por outro, os triplos correctos nem sempre têm grande utilidade, tal como discutido na secção 2. Logo, este método terá em conta do numero de ocorrências e utilizações dos vários átomos do termo composto em colecções de documentos.

Numa fase posterior do trabalho pretendemos vir a integrar um conjunto de triplos extraídos da Wikipédia, também de forma automática, numa ontologia lexical ao estilo da WordNet mas para o português. À semelhança do que foi feito por [18], os termos serão associados, ou darão origem, a synsets, e os triplos passar-se-ão a estabelecer entre synsets. Este tipo de estruturas são uma forma aceitável de lidar com a ambiguidade e, além disso, permitirão a inferência de novas relações. Uma primeira abordagem a este problema, onde são utilizados recursos lexicais para o português, é descrita em [11].

Há ainda a acrescentar que, futuramente, pretendemos disponibilizar os resultados deste trabalho para toda a comunidade que trabalhe com o processamento computacional da língua portuguesa.

Referências

1. Afonso, S., Bick, E., Haber, R., Santos, D.: Floresta sintá(c)tica: um treebank para o português. In: Gonçalves, A., Correia, C.N. (eds.) Actas do XVII Encontro Nacional da Associação Portuguesa de Linguística (APL 2001). pp. 533–545. APL, Lisboa (2001)
2. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia – a crystallization point for the web of data. Web Semantics: Science, Services and Agents on the World Wide Web 7(3), 154–165 (Setembro 2009)
3. Bollegala, D., Matsuo, Y., Ishizuka, M.: Measuring semantic similarity between words using web search engines. In: Proc. 16th International conference on World Wide Web (WWW'07). pp. 757–766. ACM, New York, NY, USA (2007)

4. Cardoso, N.: REMBRANDT - Reconhecimento de Entidades Mencionadas Baseado em Relações e ANálise Detalhada do Texto. In: Mota, C., Santos, D. (eds.) Desafios na avaliação conjunta do reconhecimento de entidades mencionadas, pp. 195–211. Linguateca (2008)
5. Cederberg, S., Widdows, D.: Using lsa and noun coordination information to improve the precision and recall of automatic hyponymy extraction. In: Proc. 7th Conference on Computational Natural Language Learning (CoNLL). pp. 111–118. Association for Computational Linguistics, Morristown, NJ, USA (2003)
6. Chodorow, M.S., Byrd, R.J., Heidorn, G.E.: Extracting semantic hierarchies from a large on-line dictionary. In: Proceedings of the 23rd annual meeting on Association for Computational Linguistics. pp. 299–304. Association for Computational Linguistics, Morristown, NJ, USA (1985)
7. Costa, H., Gonçalo Oliveira, H., Gomes, P.: The impact of distributional metrics in the quality of relational triples. In: Proc. ECAI Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH 2010) (2010), no prelo
8. Costa, L., Santos, D., Rocha, P.A.: Estudando o português tal como é usado: o serviço AC/DC. In: The 7th Brazilian Symposium in Information and Human Language Technology (STIL 2009) (2009)
9. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database (Language, Speech, and Communication). The MIT Press (1998)
10. Freitas, M.C.: Elaboração automática de ontologias de domínio: discussão e resultados. Ph.D. thesis, Pontifícia Universidade Católica do Rio de Janeiro (2007)
11. Gonçalo Oliveira, H., Gomes, P.: Towards the automatic creation of a wordnet from a term-based lexical network. In: Proceedings of the ACL Workshop TextGraphs-5: Graph-based Methods for Natural Language Processing (2010), no prelo
12. Gonçalo Oliveira, H., Santos, D., Gomes, P.: Extracção de relações semânticas entre palavras a partir de um dicionário: o PAPEL e sua avaliação. Linguamática 2(1), 77–93 (Maio 2010), nova versão, revista e aumentada, da publicação Gonçalo Oliveira et al (2009), no STIL 2009
13. Hearst, M.A.: Automated discovery of wordnet relations. In: [9], pp. 131–151 (1998)
14. Herbelot, A., Copestake, A.: Acquiring ontological relationships from wikipedia using RMRS. In: Proc. ISWC 2006 Workshop on Web Content Mining with Human Language Technologies (2006)
15. Medelyan, O., Milne, D., Legg, C., Witten, I.H.: Mining meaning from wikipedia. Intl. Journal of Human-Computer Studies (Maio 2009)
16. Oliveira, P.C.: Probabilistic Reasoning in the Semantic Web using Markov Logic. Master's thesis, Universidade de Coimbra, Faculdade de Ciências e Tecnologia, Departmento de Engenharia Informática (2009)
17. Richardson, S.D., Dolan, W.B., Vanderwende, L.: Mindnet: Acquiring and structuring semantic information from text. In: Proc. 17th Intl. Conf. on Computational Linguistics (COLING). pp. 1098–1102 (1998)
18. Ruiz-Casado, M., Alfonseca, E., Castells, P.: Automatic assignment of wikipedia encyclopedic entries to wordnet synsets. In: Proc. Advances in Web Intelligence 3rd Intl. Atlantic Web Intelligence Conference (AWIC). pp. 380–386. Springer (2005)
19. Ruiz-Casado, M., Alfonseca, E., Castells, P.: Automatising the learning of lexical patterns: An application to the enrichment of wordnet by extracting semantic relationships from wikipedia. Data Knowledge Engineering 61(3), 484–499 (2007)
20. Wandmacher, T., Ovchinnikova, E., Krumnack, U., Dittmann, H.: Extraction, evaluation and integration of lexical-semantic relations for the automated construction of a lexical ontology. In: Third Australasian Ontology Workshop (AOW 2007). CRPIT, vol. 85, pp. 61–69. ACS, Gold Coast, Australia (2007)
21. Zesch, T., Müller, C., Gurevych, I.: Extracting lexical semantic knowledge from Wikipedia and Wiktionary. In: Proc. 6th Intl. Language Resources and Evaluation (LREC'08). Marrakech, Morocco (2008)

Extraction of Family Relations between Entities

Daniel Santos, Nuno Mamede, Jorge Baptista

IST – Instituto Superior Técnico
Universidade do Algarve
L^{2}F – Spoken Language Systems Laboratory – INESC ID Lisboa
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{daniel.santos,nuno.mamede}@ist.utl.pt,jbaptis@ualg.pt

Abstract. Nowadays, there is a growing need to automatically extract information from texts. In this active research field much effort has been invested to improve the identification and classification of named entities, the detection of time expressions, and the identification of semantic relations between text entities. This paper presents a system that identifies and classifies family relations. The directives, the options used, the implementation and the results obtained are here presented.

Resumo. Existe uma necessidade crescente em extrair informação a partir de um texto, nomeadamente na identificação e classificação de entidades mencionadas e na identificação de relações semânticas entre essas entidades. Este artigo apresenta um sistema que identifica relações familiares. Descrevem-se as directivas de identificação e anotação, as opções adoptadas, a implementação do sistema e os resultados obtidos.

1 Introduction

Automatic extraction of semantic knowledge is one of the goals of Natural Language Processing. The extraction of semantic relations between entities represented in a text can improve the performance of systems that rely on this type of information, such as question/answering systems and text summarization systems. Family relations are a particularly well defined set of semantic relations. Historical and biographical documents are examples of texts that are every rich in Family Relations.

Although the evaluation of these type of systems for the English language has produced very good results, for the Portuguese language the extraction of semantic relations is still in an early phase and results are not as good yet. In the late 80's, the first major evaluation campaigns for relation extraction in the English language took place in MUC[1] (Message Understanding Conference). In the late 90's, the ACE (Automatic Content Extraction) conference promoted a joint evaluation almost every year.

For the Portuguese language, the only joint evaluation contest ever held for the extraction of semantic relations took place in 2008, as a specific track of the

[1] http://www.itl.nist.gov/iad/894.02/related_projects/muc/proceedings/ie_task.html

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 549–560

HAREM conference (Avaliação e Reconhecimento de Entidades Mencionadas) [Mota & Santos, 2008]. Results reported in this task were not as good as those achieved in the Named Entities Recognition task.

This paper describes the implementation of a system built to identify *family* relations (like parenting, sibling, etc.), for the Portuguese language and reports on an evaluation of that system. The main goal is to get a good f-measure minimizing the number of incorrect classifications, ie, maximizing precision.

The remainder of this paper is organized as follows. Section 2 presents the state of the art and the methodologies used. Section 3 describes the architecture and the implementation of this system. Section 4 presents the strategy used to extract the semantic relations, while Section 5 contains the evaluation. Finally, in Section 6 some conclusions are drawn in order to structure future work.

2 Related Work

The *Family* relation type has been present in every ACE edition, although it underwent some changes over time. In the earlier editions[2], this category was associated to several subcategories, but after the fourth edition[3] there has been only one major category *Family*, which includes all different types of family relations. In HAREM, the Portuguese joint evaluation contest, the *Family* category was also present as one of the subcategories of "Outras" [Freitas et. al., 2009].

Both evaluations consider all family relations in a single category without specifying the type of the relation. We decided that family relations should be differentiated from other types of semantic relations. In order to do that, we created a set of features to identify this particular type of semantic relation.

Two major groups of methods are usually adopted for the semantic relations extraction task: rule-based and machine learning approaches. All systems evaluated in HAREM are based on rules, i.e, these systems analyze the syntactic structure of sentences looking for patterns and then, based on the information that is present on every sentence and the patterns extracted, they deduce relations among entities. For example, in the sentence *O João é primo do José* "João is José's cousin" if the system matches the pattern consisting in a noun phrase (NP) whose head is a human noun (e.g. *João*), the verb *ser* "to be", a family relation noun (e.g. *primo* "cousin") and a prepositional phrase (PP) introduced by *de* "of" with another human head noun (e.g. *José*); and if a subject relation has been established between the first NP and the verb, while a modifier relation has been found to exist between the head of the last NP and the family relation noun, then a *Family* relation between the two human nouns should be established. The best system in the global task of relation extraction in HAREM was REMBRANDT [Cardoso, 2008] which is based on a set of rules used to infer

[2] More information about ACE02 in *"ACE Evaluation plan version 06"* is available in `http://www.itl.nist.gov/iad/mig/tests/ace/2002/doc/`

[3] *"Version (7) of the 2004 ACE evaluation plan"* in `http://www.itl.nist.gov/iad/mig/tests/ace/2004/`

new relations. The result achieved was 45% f-measure, with 60% precision and 35% recall.

Culotta and Sorensen [Culotta & Sorensen, 2004] use the same categories defined in the first two ACE editions, and they try different approaches for relation extraction. They created a system based on machine learning with dependency tree kernels. The first step consists in building a parse tree using a maximum entropy statistical parser. This tree is then converted to a dependency tree that represents the grammatical relations between words in a sentence. The final step consists on the application of kernel methods, defined by the authors, for the extraction of relations among entities. They report a best result of 63.2% f-measure, with 81.2% precision and 51.8% recall.

3 Architecture

In this paper, a rule-base approach is adopted, since there is still no available corpus for the Portuguese language annotated for family semantic relations. The corpus used in HAREM cannot be used because: the annotations did not include different types of family relations, the number of annotations in the data set is too small for an efficient use of machine learning methods, and finally, the annotations in this corpus are made between named entities and in this paper we extract relations between any type of entities.

Furthermore, because semantic relations can be viewed as another layer of information over syntactic dependencies, already being extracted by the syntactic parser here used, it would be easier to extend the rule-based grammar already implemented in this system [Mamede, 2007] to encompass also semantic relations.

The identification of relations between entities is thus performed in XIP (Xerox Incremental Parsing), one of the modules of the L2F[4] NLP processing chain, whose structure is sketched in Figure 1, and that will be briefly presented below.

The first module (Segment Splitter) splits the text into tokens, while Palavroso [Medeiros, 1995] assigns to each token all the possible part-of-speech (POS) tags, depending on the token ambiguity.

The Sentence Splitter splits the text into sentences. Every time the system finds one of the following characters ".", "!" and "?" it considers it as the end of the current sentence. The result is converted into an adequate format and piped into RuDriCo [Pardal, 2007], which uses several heuristics to remove or select some of the POS that were given by the morphosyntactic labeling module. This type of rules is based on previously known cases and they choose or eliminate some specific POS for a given token given its neighboring context. Another functionality of RuDriCo is the joining of strings of words forming compounds as single tokens and splitting of contracted word forms into their component words; for instance, the words *Coreia*; *do*; *Norte* becomes a single token *Coreia do Norte*

[4] Laboratório de Sistemas de Língua Falada do Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento

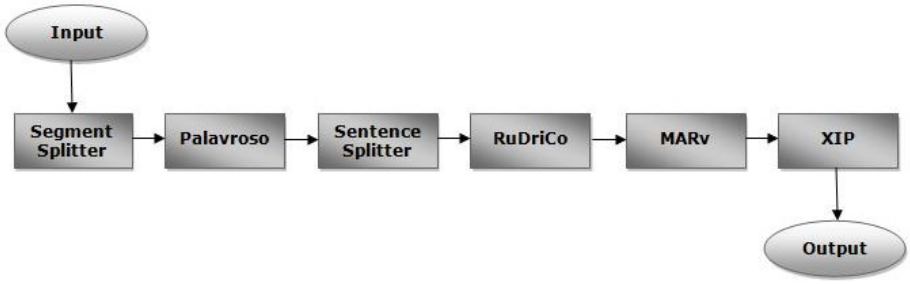


Fig. 1. L2F NLP Processing Chain.

("North Corea"), while the contraction *disso* is split in to preposition *de* ("of") and pronoun *isso* ("that"). The MARv module [Ribeiro et. al., 2003] performs the remaining disambiguation. It uses a statistical model and the Viterbi algorithm to choose the most probable category for each given token in the sentence. The result is again converted and piped into XIP [Xerox, 2003], where a set of complex operations is carried out, namely the structuring of the sentences into chunks, the extraction of syntactic dependencies, named entities recognition, and anaphora resolution. It is also at this stage that the extraction of semantic relations takes place.

4 Methodology

Only one major category, named *FAMILY*, has been defined, covering all family relations. Each relation will be associated to a feature that expresses the relation type (uncle, parenting, sibling, etc.) and the gender of the relation arguments. For example, in the sentence *O João é primo do José* "João is José's cousin", already presented above, the following semantic relation is extracted:

```
FAMILY_cousin_1M_2M(João,José)
```

To look for relevant patterns, the words (mainly nouns) that express family relations are very important lexical clues. For example, the word *pai* ("father") is a good lead, but its mere presence does not mean that we have a family relation in that sentence, for example in sentences like *São Pacómio, pai da vida monástica cenobítica*[5] "St. Pachomius, the father of cenobitic monastic life", where *pai* is used in the sense of "founding father". In order to solve this and similar problems, rules have to be rendered much more precise to ensure that a family relation is captured only if its arguments are human nouns. These include proper nouns, professions, titles, or generic human nouns like, man, woman, children, etc.

In order to identify family relations, a survey of the syntactic patterns that these relation nouns determine was carried out. Some of these patterns are similar regardless of the semantic relation they express.

[5] `http://hagiaecclesia.blogspot.com/2009/05/sao-pacomio-c.html`.

For example, the following sentences: *O João é pai do Pedro* "John is Peter's father", *O João é tio do Pedro* "John is Peter's uncle", and *O João é irmão do Pedro* "John is Peter's brother" all have the same syntactic structure: *João* is always the subject, the verb *ser* "to be" links the subject to the noun referring the type of relation, and the `PP` with *Pedro* is governed by the relation noun. Instead of making a specific rule for each relation, a general rule is constructed in order to capture all these cases, while other rules specify the relation type present in each sentence.

Next, before detailing how these rules were built, a brief overview of the XIP syntax and of the dependency types used to extract family relations in presented. XIP syntax is based on regular expressions. For clarity, XIP rules are split in three parts (all of these three parts are optional):

```
|pattern| if <condition> <dependency terms>.
```

- The `pattern` part regards the nodes of a given sentence. A node or a chunk is composed by one or more words, like a noun, a verb, an article, a preposition, etc. The most common nodes are `NP` (noun phrase), `VP` (verb phrase), `PP` (prepositional phrase). It is also possible to verify the presence of some word features, for example the gender (masculine or feminine), and the number (singular or plural). The features used in the extraction of family relations are related to gender, number, the lemma of a word, the feature "relative" that is present in every word related to a family relation (father, mother, uncle, brother, sister, etc.) and finally we verify if a noun may represent a person through the presence one of these features: people, individual, human or profession.
- The `<condition>` part is an if clause, which is used to verify some conditions, like the presence of a dependency that conveys a some specific meaning in the sentence. For example the dependency `SUBJ` identifies the subject of the verb; for the sentence *O João é irmão do Pedro* "John is Peter's brother" the subject dependency is created: `SUBJ(é,João)`.
- Several dependencies are used in the relation extraction task:
  - The `PREDSUBJ` dependency links a copula verb like *ser* "to be" to a predicative noun, an adjective or adverb; for example, in the sentence, *O João é irmão do Pedro* "João is Pedro's brother", we have a `PREDSUBJ` dependency between the verb *ser* "to be" and the noun *irmão* "brother".
  - The `APPOSIT` dependency links the noun with an apposite; in the sentence *O João, o irmão do Pedro, fez isso* "João, Pedro's brother, did that", the following dependency is extracted: `APPOSIT(João,irmão)`.
  - The coordination dependency `COORD` links elements in a coordination chain. For instance, if a verb operates on a noun and that noun has a coordination dependency with another noun then the second noun is also operated upon by the same verb (and is in the same syntactic relation to the verb as the first noun).
  - The `HEAD` dependency relates the nucleus of a chunk with the chunk itself; in the previous sentence, the noun *João* is the head of the nominal chunk *O João*.

– Finally, the `<dependency terms>` determines the action of the rule. In the relation extraction task, this part of the rule creates the family dependency.

The rule presented in Figure 2 is used to determine if a sentence has a family relation. The first part imposes restrictions on the tree structure of the sentence: this must have a `NP` that is composed by something (?*) and a noun presenting semantic, human-related features (people and individual, human, people or profession); this `NP` can be followed by some optional `NP` or `PP` (these elements often indicate the age or the profession of the first `NP`; for the kind of relations here targeted, that information is not relevant); after those optional chunks there must be a `VF` with a verb whose lemma is *ser* "to be"; Then we have another `NP` with a noun that has the "relative" feature (like "father", "uncle", "brother"); finally, we have again a `PP` with a noun referring to a human.

Afterwards, we verify some conditions by way of an if clause. The noun in the first `NP` has to be the head of the chunk, the same thing happens in the `PP`. The head of the first `NP` must be the subject of the verb, and the noun of the second `NP` has to be in a `PREDSUBJ` relation with the verb.

At last, and if no previous family relation has been detected, we extract the `FAMILY` relation between the two human nouns and in this first phase we keep the type of the relation that is present in the second `NP`.

```
| NP#1{?*, noun#2[people, individual]; noun#2[human];
          noun#2[people]; noun#2[profession]},
  PUNCT*, NP*, PP*, PUNCT*,
  VF{verb#3 [lemma:ser]}, NP{noun#4[relative]},
  PP#5{?*,noun#6[people, individual]; noun#6[human];
         noun#6[people]; noun#6[profession]} |

 if( HEAD(#2,#1) & HEAD(#6,#5) & PREDSUBJ(#3,#4) &
     SUBJ[PRE](#3,#2) &
     ~FAMILY(#2,#6) & ~FAMILY(#4,#2,#6))

  FAMILY(#4,#2,#6)
```

Fig. 2. XIP rule: creates a new Family relation

Next, as shown in Figure 3, a set of rules is used to remove the type of relation from the first argument of the `FAMILY` dependency and to add the relation type as a feature of the dependency, now with only two arguments. In the if clause we erase the previous dependency created and verify which is the lemma for the first argument, e.g.*primo* "cousin"; if that is the case, then we create a new dependency `FAMILY` with the feature `cousin` included. For each different lemma referring a family relation, a similar rule has to be made.

It is also necessary to identify the gender of both arguments in a family relation. In order to do that four features have been created: `1M`; `2M`; `1F` and `2F`. These features indicate whether the first or the second argument is either

```
if( ^FAMILY(#1,#2,#3) & #1[lemma:primo])
   FAMILY[cousin=+](#2,#3)
```

Fig. 3. XIP rule: remove the type of relation from the arguments.

masculine or feminine. Figure 4 contains an example of this type of rules. In this rule the dependency `FAMILY` is checked to determine if it does not have the gender feature already, and if the first argument has the feature `masc` (masculine) and does not have the feature `fem` (feminine). If these conditions are satisfied, then the feature `1M` is added, indicating that the first argument is masculine.

```
if( ^FAMILY(#1,#2) & ~FAMILY[1M](#1,#2) & ~FAMILY[1F](#1,#2) &
     #1[masc] & ~#1[fem])
    FAMILY[1M=+](#1,#2)
```

Fig. 4. XIP rule: add the gender feature to the relation.

Although these rules identify most of the cases, some given names are ambiguous and all family names are not marked for gender [Baptista et. al., 2006]. In those cases we decided not to include the gender of that argument.

To solve some of the ambiguity present, some rules were further refined. These take into account the fact that often the noun expressing the relation type indicates the gender of one of its arguments. For example in the sentence *Saraiva é tio de Silva* "Saraiva is Silva's uncle" we know that the family proper name *Saraiva* represents a male person because of the relation noun here used.

We have also removed some ambiguity in nouns by changing its gender based on the article that usually precedes the noun. Notice that in Brazilian Portuguese the article is usually not used, while in European Portuguese the presence or absence of the article is meaningful and its use is related to the degree of notoriety or the familiarity of the speaker with the individual.

Several idiomatic expressions also convey familiar relations, for example the sentence *O João e a Joana deram o nó* (literally, "João and Joana have tied the knot") means that these two people got married. These cases are different from the ones we presented before because, instead of a global rule, each one must have a specific rule since it conveys a single relation type and have a specific syntactic structure.

Many relation nouns can be used in combination with other lexical elements in order to distinguish or to define in a more specific way a basic family relation; for example adoptive father, foster father, twin brother, etc. To handle these type of relations we use regular expressions in the lemmas. For example, if the lemma: "pai" is used, this expression will only match the word *pai* "father", however if we use as lemma: "pai(%c*)" then it will also match *pai adoptivo* "adoptive father".

Symmetric relations, like *irmão* "sibling", *cunhado* "brother-in-law" and *primo* "cousin" also require specific rules to deal with sentences such as *João e o Pedro são primos* "João and Pedro are cousins", which are not captured by the general rules. The symmetry property consists in the arguments of the relation noun being able to appear coordinated in the subject position. Non-symmetric relation nouns cannot enter this syntactic pattern: *João e o Pedro são pais* "João and Pedro are fathers" (this sentence would be acceptable but the "father" relation would not hold between the two human nouns).

So far, all relations presented here have two arguments, but some expressions may appear to have more than two, for example, the sentence *O João e o Carlos são tios do Pedro* "João and Carlos are Pedro's uncles".

In this sentence, the family relation "uncle" holds not only between *João* and *Pedro*, but also between *Carlos* and *Pedro*. In order to capture cases like this, where there is more than one relation in the same sentence, the following rules were made:

```
if( FAMILY(#1,#2,#3) & #1[pl] & COORD(#4,#2) & COORD(#4,#5))
   FAMILY(#1,#5,#3)

if( FAMILY(#1,#2,#3) & COORD(#4,#3) & COORD(#4,#5))
   FAMILY(#1,#2,#5)
```

Fig. 5. XIP rule: creates an additional relation in cases where a sentence has a relation with three arguments.

These rules verify if there is a coordination dependency between one of the arguments of the previously detected relation and another entity. Whenever this condition is met the same relation is propagated to the second entity.

The last special case is related to anaphora. Anaphora may be defined as the referential relation that holds between two instances in a text: an expression (the *anaphor*) that refers to another expression, which has occurred previously in the same text (the *antecedent*). A module for anaphora resolution is currently being developed for the Portuguese language at L2F/INESC-ID by another researcher. Once this module is in place, its results are likely to improve the relations extraction task.

Among the different types of anaphoric devices, zero anaphora [Mitkov, 2002] constitutes a particularly challenge to anaphora resolution systems. Zero anaphora holds between a void anaphor, i.e. an empty syntactic slot, and its antecedent; it is a form of ellipsis, used to avoid word repetition.

For example, in the sentence *O João é irmão do Pedro mas cunhado do Carlos* "João is Pedro's brother but [he is] Carlo's brother-in-law" the subject of the relation noun *cunhado* is also *João* but it has been zeroed not to be repeated, since it refers to the subject of the first coordinate sentence [Pereira, 2010].

Zeroing also occurs in discourse following turn taking (e.g. answering a question), like in the sentence: *É tio do Pedro* "[He] is Pedro's uncle". In these cases,

the zeroed anaphor may be identified but its reference can not be solved at this stage, for its antecedent is not in the current sentence. Therefore, a dummy node is created, inheriting the features that the relation extraction rules can recover from the context (v.g. in the sentence above, the masculine, singular, third person). In the extracted relation, a dummy feature *A0* is added. At a future stage of the anaphora resolution module, this information would be used to correctly calculate the antecedent of this zero anaphor.

5 Evaluation

For the evaluation of this task, we use three metrics which are common to other research papers in this field. These three metrics are:

$$Precision = \frac{Correct\_Relations}{Relations\_Identified}$$

$$Recall = \frac{Correct\_Relations}{Total\_Relations}$$

$$f - measure = \frac{2 * precision * recall}{precision + recall}$$

At this stage, we only consider for evaluation purposes the patterns where the relation noun and its argument named entities are explicitly present in the text, such as in the examples discussed above. Other cases will be discussed in the final section of the paper. We have implemented 99 rules to extract these family relations.

Since the *Family* category has been treated differently by the systems presented in Chapter 2, and also because the evaluation corpus is different in almost every investigation (except in joint evaluations), it is not possible to compare rigorously the system here presented with those referred to above.

Two evaluation corpora were used to evaluate the relation extraction task. The first evaluation corpus is a text containing the biography of all Portuguese kings. We decided to use this type of documents because they are very rich in family relations and they have common Portuguese names that should be easily identified as a person named entities by the system. These biographies were gathered from Wikipedia[6] and they were then manually annotated for the family relations they present.

The total number of family relations present in this evaluation corpus is 105. The annotation task was made after the implementation phase, so that we could not adapt our rules to these specific cases.

We performed a second evaluation because we noticed that the Portuguese Kings biographies have many implicit relations, assuming that whoever is looking at the text already knows something about the person whose life is being described. An automatic system does not work so, and this may influence the results.

[6] `http://www.wikipedia.pt`

The second evaluation corpus is then made up of the first 110 sentences containing at least one relation noun, from a list of about 100 names, and retrieved from the CETEMPúblico corpus[7].

The results of the first evaluation corpus are presented in Table 1. Some factors hindered these results, namely the implicit relations, mentioned above, false positive relations, identification of relations with the incorrect arguments, and other errors due to an incorrect performance of the NER module, which is beyond the scope of this task.

For example, in the following sentence: *O acordo foi firmado em 1174 pelo casamento de Sancho, então príncipe herdeiro, com a infanta Dulce Berenguer, irmã mais nova do rei Afonso II de Aragão.* The relation that should have been extracted is:

```
FAMILY_SPOUSE(Sancho, infanta Dulce Berenguer)
```

But the relationship that has actually been extracted is:

```
FAMILY_SPOUSE(Sancho, infanta)
```

The problem in this example is that a composite node between the title *infanta* "princess" and *Dulce Berenguer* should have been created; if it had, the relation would be correctly established. Therefore, results were reassessed counting cases like this as correct, in order to see their impact on the overall results. As seen on the right side of Table 1, these results are indeed better.

Table 1. Results of the first evaluation corpus (the right table considers as correct the relations with errors on the identification of people names).

Precision	Recall	F-measure
0.59	0.19	0.29

Precision	Recall	F-measure
0.71	0.23	0.35

Table 2 presents the results for the second evaluation corpus. This second corpus contained 110 sentences; only 21 of these had an explicit family relation, while the remaining 89 contained at least one relation word, but no explicit family relation to associate pairs of entities. As was mentioned above, only explicit relations with both entities expressed were considered for evaluation in this paper.

Table 2. Results of the second evaluation corpus

Precision	Recall	F-measure
0.71	0.24	0.36

[7] `http://www.publico.pt/`

Some of the cases that lowered the precision are due to the incorrect identification of the arguments, i.e., the relation is explicitly present in the sentence, but the rule is unable to correctly capture one of the arguments. The following example exhibits this problem:

Quanto ao Troféu BMW 320iS, Jorge Petiz ultrapassou o seu irmão Alcides a meio da corrida para obter uma vitória fácil, com 2,382″ de avanço, deixando o 3o, António Barros, a 4,788″.

In this case, the name *Alcides* is an apposite to the word *irmão ("brother")*; only this half of the relation (*FAMILY_A0_1M_SIBLING*) is correctly identified but a dummy (anaphoric) first argument (A0) is construed since the system is currently unable to identify *Jorge Petiz*, the sentence subject, as the relation's first argument. In order to do so, the reference of the possessive pronoun would have to be resolved, which is outside the scope of this paper.

The evaluation corpus also contains some foreign names that are not identified as person named entities by the system, thus preventing the relation extraction. This problem derives from the NER module and not from the relation extraction module. Therefore, it was decided to add the missing names to the XIP's lexicon, and to perform a new evaluation. As expected the results are better (see Table 3), particularly the recall.

Table 3. New Results of the second evaluation corpus

Precision	Recall	F-measure
0.70	0.33	0.45

6 Conclusions and future work

The extraction of semantic relations between named entities in text is a very complex and challenging task. This paper reports a first attempt to extract from texts semantic relations between named entities, using the NLP chain built at L2F. Results from two different corpora are still unsatisfactory. The main problems detected came from (i) the insufficient performance of the named entity recognition module, which produced much of the incorrect matching of the relations' arguments; and (ii) the limited coverage of the syntactic-semantic dependency extraction module (deep parser). Several dependency rules were built to capture new patterns in order to improve the recall measure of the task.

One of the dependencies that requires further attention is apposition, as in the sentence: [. . .]*o grupo tinha em seu poder o tenente-coronel Mike Couillard, 37 anos, e o seu filho Matthew, dez anos.* "the group had in his power the lieutenant-coronel Mike Couillard, age 37, and his son Matthew, age 10". Even if the age insertions were sorted out, the system still does not correctly handle titles, when they are adjoined to a proper name. The same happens with relation nouns, like *filho* "son", often appearing in front of the named entity.

Data from the corpora have shown that family relations in texts are most often expressed by way of incomplete mention, using just the relation noun, as in the following example: *No regresso, o meu pai já vinha a dormir.* "On the way home, my father was already sleeping". In this case, a parenting relation is present, but the relation noun also designates a person whose reference still needs to be established in the previous discourse. On the other hand, the possessive refers to the enunciation subject, which calls for a much complex calculation, across direct and reported speech. Correference resolution is therefore an unavoidable track parallel to relation extraction task.

References

[Baptista et. al., 2006] Jorge Baptista, Fernando Batista, Nuno Mamede, Building a Dictionary of Anthroponyms, In PROPOR 2006 - Computational Processing of the Portuguese Language, Springer Verlag, Berlin Heidelberg, vol. 3960, pages 21-30, Itatiaia, Brazil, May 2006.

[Cardoso, 2008] Nuno Cardoso: Rembrandt - reconhecimento de entidades mencionadas baseado em relações e análise detalhada do texto, Chapter 11, pages195-211, 2008.

[Culotta & Sorensen, 2004] Aron Culotta, Jeffrey Sorensen: Dependency tree kernels for relation extraction. In ACL'04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, page423, Morristown, NJ, EUA, 2004. Association for Computational Linguistics.

[Mamede, 2007] Nuno Mamede: A Cadeia de Processamento XIP. L2F (Laboratório de Sistemas de Língua Falada, Maio 2007.

[Medeiros, 1995] J. C. Medeiros, Processamento Morfológico e Correcção Ortográfica do Português. Master's thesis, Instituto Superior Técnico - Universidade Técnica de Lisboa, Portugal, 1995.

[Mitkov, 2002] Ruslan Mitkov, Anaphora Resolution, Pearson ESL, 1st Edition, 2002, ISBN: 978-0582325050, Chapter 1.

[Mota & Santos, 2008] Cristina Mota, Diana Santos: Desafios na avaliação conjunta do reconhecimento de entidade smencionadas: O Segundo HAREM, 2008.

[Freitas et. al., 2009] Cláudia Freitas, Diana Santos, Cristina Mota, Hugo Gonçalo Oliveira, Paula Carvalho: Relation detection between named entities: report of a shared task. In DEW'09: Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions, pages 129-137, Morristown, NJ, EUA, 2009. Association for Computational Linguistics.

[Pardal, 2007] Joana Pardal, Manual do Utilizador do RuDriCo. Technical report, Instituto Superior Técnico - Universidade Técnica de Lisboa, Portugal, 2007.

[Ribeiro et. al., 2003] Ricardo Ribeiro, Nuno Mamede, and Isabel Trancoso, *Computational Processing of the Portuguese Language: 6th International Workshop, PROPOR 2003, Faro, Portugal, June 26-27, 2003*, volume 2721 of Lecture Notes in Computer Science. chapter Using Morphossyntactic Information in TTS Systems: Comparing Strategies for European Portuguese. Springer, 2003.

[Pereira, 2010] Simone Pereira, Linguistic Parameters for Anaphora Resolution, MA thesis, Univ. Algarve, Faro, 2010.

[Xerox, 2003] Xerox, Xerox Incremental parser – Reference Guide, 2003.

O impacto de diferentes fontes de conhecimento na marcação de Nomes Próprios em Português

João Tomé da Silva Laranjinho and Irene Pimenta Rodrigues

joao.laranjinho@gmail.com, ipr@di.uevora.pt
Departamento de Informática
Universidade de Évora, Évora, Portugal

Resumo Neste artigo apresenta-se um sistema, independente do domínio, para marcação de nomes de entidades para o português. Este sistema é avaliado de forma a estudar o impacto de diferentes fontes de conhecimento nos resultados do sistema.
O marcador usa informação morfo-sintáctica, sintáctica e semântica. A informação morfo-sintáctica vem de um dicionário local que completa a sua informação recorrendo a dicionários disponíveis na rede como o da Priberam. A informação sintáctica das frases vem de uma gramática construída para analisar frases interrogativas, esta gramática tem bons resultados para as frases interrogativas (acima de 95% de cobertura) mas para as frases dos outros corpura testados tem um mau desempenho (abaixo dos 30% de cobertura). A informação semântica usada nas experiências de avaliação vem da Wikipédia.
Na avaliação do sistema e do impacto das diferentes fontes de informação usaram-se três corpura distintos: um conjunto de documentos com 700 perguntas do CLEF; 300 frases de noticias do Público; e 200 frases do corpus CD do segundo Harem.

Abstract We present a domain independent system that identifies proper names in Portuguese texts. This system is evaluated in order to study the impact of the different knowledge sources on its performance. The knowledge sources are: morph-syntactic obtained from a local dictionary that is able to consult online dictionaries; syntactic from a Portuguese grammar that indicates if a sentence with some proper names tagged is correct; and semantic obtained from an online encyclopedia, the Wikipedia.
In the evaluation, we use three different corpus: a set of documents with 700 questions from CLEF, 300 sentences of news from Publico, and 200 sentences from the corpus of the second Harem. The Portuguese grammar was developed in order to analyse Portuguese interrogative sentences. This fact is reflected in the performance of our system in the different corpus since the coverage of the grammar is up to 95% for CLEF and bellow 30% for the others.

1 Introdução

Os sistemas de marcação de nomes de entidades são um caso particular (a identificação) dos sistemas de reconhecimento de entidades mencionadas (REM) que identificam e classificam as entidades de acordo com uma hierarquia pré-definida de categorias. Para

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 561–572

o inglês, alguns dos sistemas actuais conseguem um valor de 93% na medida F em foruns de avaliação como o MUC-7 [NS07]. Para o português, no segundo Harem, o melhor sistema, o sistema da Priberam [AFM+08] consegue atingir cerca de 71% para a medida F na identificação de entidades mencionadas (ou marcação de nomes de entidades).

O sistema REMUE (Reconhecimento de Entidades Mencionadas da Universidade de Évora) foi, inicialmente, desenvolvido para melhorar o desempenho de um sistema de resposta automática a perguntas [QRPV06], em particular para auxiliar na escolha das melhores análises sintácticas de uma pergunta em português. A avaliação inicial do REMUE foi feita usando um corpus com as perguntas do CLEF na tarefa de pergunta-resposta [SR04] de várias edições. A marcação dos nomes de entidades neste corpus foi feita manualmente para a avaliação do REMUE. Para estudar o comportamento do REMUE noutro tipo de corpus construiu-se uma amostra com frases do Público que também foram anotadas manualmente. E finalmente fez-se uma avaliação com uma amostra da colecção dourada do Segundo Harem que apesar de não permitir uma comparação directa com os sistemas que concorreram ao Harem nos permite ver o impacto das diferentes fontes de conhecimento nos resultados do REMUE (o REMUE não foi avaliado no Segundo Harem).

O REMUE usa três tipos de fontes de conhecimento para decidir a marcação de nomes de entidades:

- Informação morfo-sintáctica - num dicionário local que completa a sua informação recorrendo a dicionários disponíveis na rede como o Dicionário da Priberam.
- Informação sintáctica - o resultado de um analisador sintáctico, a estrutura sintáctica das frases com os nomes de entidades das frases. Uma frase pode ter zero, uma ou mais estruturas sintácticas associadas. A estrutura sintáctica pode ser parcial ou total. Esta informação é usada, pelo REMUE, para decidir a melhor marcação de nomes de entidades na frase.
- Informação semântica - informação de dicionários e enciclopédias que indicam se o nome da entidade existe nalgum contexto. No REMUE por enquanto só se usa a informação da Wikipédia que como se pode ver na sua avaliação tem um grande impacto na correcção das marcações.

As técnicas usadas nos sistemas de REM dividem-se em: modelos baseados em regras e modelos estatísticos. Actualmente a técnica dominante é a baseada em modelos estatísticos e aprendizagem. Estes sistemas requerem grandes quantidades de dados anotados manualmente para a fase de treino e alguns sistemas tem um desempenho dependente do domínio.

Os modelos baseados em regras têm apresentado melhores resultados (ver MUC-7), no entanto, requerem muito trabalho feito por linguistas na sua implementação e muitas vezes o seu desempenho depende do domínio.

Algum dos recursos utilizados no REM [AA08] são:

- Corpus - conjuntos de textos anotados. Normalmente em conjunto com os corpura, são utilizadas estratégias de aprendizagem, como por exemplo: modelos de Markov não observáveis (Hidden Markov Models - HMM), árvores de decisão, modelos de

máxima entropia, SVMs [NS07]. Um dos problemas da construção de um corpus está relacionado com a anotação;
- Almanaques - dicionários de entidades mencionadas (EM);
- Metapalavras - representam as palavras próximas das EM. Estas são palavras que dão alguma informação sobre as entidades, normalmente são utilizadas na fase de desambiguação. Exemplos: escritor, rua, etc;
- Abreviaturas - palavras que fazem parte das entidades e dão informação sobre a classe da entidade. Normalmente são utilizadas na classificação. Exemplo: Dr., Sr., etc;
- Regras de similaridade - um conjunto de regras que definem semelhanças entre as entidades a serem classificadas e as entidades que existem em almanaques.

O desempenho de um sistema de REM pode ser medido com diversas métricas [SC07] que representam o desempenho em valores numéricos.

As três métricas que normalmente são utilizadas para avaliar o desempenho de um sistema de recolha de informação são as seguintes: abrangência (Recall), precisão (Precision) e medida F (F-Measure).

- A abrangência mede a relação entre o número de resultados correctos e o número de resultados existentes. A fórmula da Abrangência é a seguinte:
  Abrangência = $\frac{Resultados\ Correctos\ \cap\ Resultados\ Existentes}{Resultados\ Existentes}$
- A precisão mede a relação entre o número de resultados correctos e o número de resultados obtidos. A fórmula da Precisão é a seguinte:
  Presisão = $\frac{Resultados\ Correctos\ \cap\ Resultados\ Obtidos}{Resultados\ Obtidos}$
- A medida F é uma média harmónica entre a precisão (P) e a abrangência (A). A fórmula da *medida F* é a seguinte:
  Medida-F = $2 * \frac{P * A}{P + A}$

Na próxima secção, 2, apresenta-se a arquitectura do REMUE com os seus módulos de processamento e as diferentes fontes de conhecimento. Na secção 3, apresentam-se os testes feitos com o REMUE em 3 corpura diferentes procurando ver o impacto das diferentes fontes de conhecimento em cada um dos corpura. Para estudar o impacto fazem-se 8 testes diferentes calculando a precisão, a cobertura e a medida F para cada corpus. Finalmente, na secção 4, analisam-se os resultados da avaliação feita e discutem-se alguns aspectos que podem ser melhorados no REMUE e na sua avaliação.

2 Arquitectura do sistema do REMUE

O REMUE recebe um ficheiro de texto para marcação de nomes de entidades e retorna dois ficheiros, um com o texto em que os nomes de entidades surgem marcados e um outro com a lista de nomes de entidades de cada frase.

Para a marcação dos nomes de entidades recorre-se a regras que codificam as preferências na escolha dos nomes de entidades. As regras usam informação sobre:

- O número de átomos do nome de entidade;

– Informação sobre a primeira letra das palavras (maiúscula ou minúscula);
– A classe morfo-sintáctica de cada palavra;
– Informação sobre alguns caracteres como: aspas, números e sinais de pontuação.

O REMUE contém 4 módulos. Na Figura 1 é apresentada a arquitectura do REMUE.

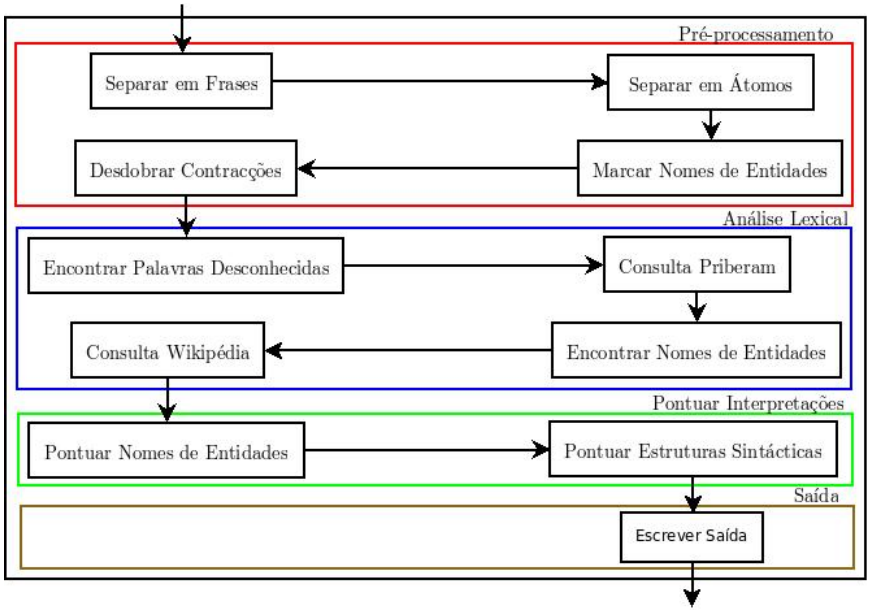


Figura 1. Arquitectura do REMUE

Os módulos são: pré-processamento, análise lexical, pontuar interpretações e saída.

2.1 Pré-processamento

Um texto para ser analisado deve encontrar-se separado em frases. Uma frase é constituída por palavras, números, sinais de pontuação, etc.

Na separação de um texto em frases utiliza-se uma estrutura que guarda, em cada posição, o conjunto de interpretações de uma frase.

Uma interpretação de uma frase para ser analisada deve encontrar-se separada em átomos. Um átomo é constituído por uma sequência de caracteres sem espaços em branco. Os átomos podem ser palavras, números, sinais de pontuação, sequências de letras alternadas com números, etc.

Um candidato a nome de entidade é constituído por uma ou várias palavras que começam com maiúscula, podendo possuir números e elementos de ligação de nome de

entidades. Os elementos de ligação de nome de entidade considerados são os seguinte: “de”, “da”, “do”, “das”, “dos”, “e”, “&” e “-”.

O sistema para uma interpretação que tem N candidatos a nomes de entidades produz 2^N interpretações. As diferenças entre às interpretações são ao nível do número de candidatos a nomes de entidades marcados que variam entre 0 e N.

2.2 Análise Lexical

Na análise lexical reúne-se num ficheiro as palavras que não se encontram no dicionário local.

O dicionário local contém informação (entradas) sobre palavras e nomes de entidades, sendo a informação sobre as palavras relativa às classes gramaticais, além de valores para variáveis morfo-sintáctico-semânticas. Enquanto que a informação sobre os nomes de entidades é relativa a sua existência.

As entradas no dicionário local são as seguintes: adjectivo (adj), advérbio (adv), determinante (det), substantivo comum (nome), número (num), preposição (prep), contracção (contr), conjunção (conj), interjeição (interj), pronome (pron), verbo (verbo) e nome de entidade (nomeEntidade).

O Priberam[1] é um dicionário disponível na rede, no qual podem ser feitas consultas sobre palavras através da Internet. Em cada consulta obtém-se informações morfo-sintáctico-semântica acerca da palavra pesquisada.

Para consultar, o Priberam foi desenvolvida uma aplicação que estabelece a ligação ao mesmo. A aplicação recebe o ficheiro de texto com as palavras. Para cada palavra estabelece-se uma ligação ao Priberam obtendo-se um ficheiro com a informação da palavra.

A informação da palavra no Priberam consiste numa tabela. Cada tabela contém duas partes (2 tr's). Na primeira parte encontra-se informação directa sobre a palavra em pesquisa. Na segunda parte além de existir informação directa, existe também informação indirecta. A informação indirecta é complementada com ligações para outras palavras que se encontram relacionadas com a palavra em pesquisa.

Por exemplo, informação que se encontra no Priberam sobre a palavra “assassino”, é a que se encontra na Tabela 1.

Relativamente à tabela com a informação da palavra “assassino”, na primeira parte da tabela é refido que esta palavra é um ajectivo masculino e um substantivo comum de género masculino. Na segunda parte é referido que a palavra “assassino” é um verbo que se encontra na 1ª pessoa do singular do presente do indicativo. Ainda na segunda parte da tabela é feita referência ao verbo “assassinar” que é o infinitivo da conjugação “assassino”.

Para processar a informação do Priberam foi criado um analisador sintáctico. O analisar sintáctico por cada pesquisa cria uma estrutura com o nome da palavra pesquisada e uma lista com as características da mesma.

As estruturas obtidas no processamento da informação de Priberam devem ser convertidas em entradas para o dicionário local, podendo uma estrutura dar origem a uma ou mais entradas.

[1] http://priberam.pt/dlpo/dlpo.aspx

```
<xml search="assassino">
  <table style="background-color:\#eee; width:100%;"
    cellpadding="4"
    cellspacing="0"
    border="0"
    bordercolor="\#cccccc">
    <tr>
      <td>
        <div>
          <b>assassino</b> |<em>adj.</em> |<em>s. m.</em>
        </div>
      </td>
    </tr>
    <tr>
      <td>
        <div>1ª pess. sing. pres. ind. de
          <a href="&#xA; default.aspx?pal=assassinar">assassinar</a>
        </div>
      </td>
    </tr>
  </table>
</xml>
```

Tabela 1. Informação da palavra "assassino" no dicionário da Priberam

A Wikipédia[2] é uma enciclopédia, na qual podem ser feitas consultas via Internet sobre nomes de entidades e outras palavras. Esta enciclopédia pode ser utilizada para verificar se determinado nome de entidade existe.

A informação sobre a existência de uma entrada para um nome de entidade na Wikipédia é feita em 2 passos.

No primeiro passo é estabelecida uma ligação à Wikipédia, que verifica se existem entradas para o nome da entidade, devolvendo-se um valor Booleano, "true" ou "false", consoante tenha ou não entrada na Wikipédia.

No segundo passo gera-se a entrada para o dicionário local, no caso de existir entrada do nome da entidade na Wikipédia (devolvido "true").

2.3 Heurística para Pontuar Interpretações

A marcação dos candidatos a nomes de entidades numa frase pode produzir várias interpretações. A melhor interpretação de uma frase é aquela que contém o maior número de nomes de entidades marcados correctamente e que pode ser representada por uma estrutura sintáctica (interpretação com análise sintáctica). Para encontrar a melhor interpretação de uma frase pode recorrer-se a uma função heurística que pontua cada interpretação.

[2] http://pt.wikipedia.org/wiki/Página_principal

Nos candidatos a nomes de entidades existem características comuns. Essas características podem ser utilizadas para agrupar os candidatos com características iguais e atribuir-lhes iguais pontuações.

Os candidatos a nomes de entidades foram divididos em:

- NE WIKI - candidato a nome de entidade que tem entrada na Wikipédia;
- NE SIMPLES - candidato a nome de entidade que não tem entrada na Wikipédia e, é composto por apenas uma palavra das seguintes classes gramaticais: substantivo comum, adjectivo ou nome próprio;
- NE COMPOSTO - candidato a nome de entidade que não tem entrada na Wikipédia e, é composto por mais que um átomo, em que o primeiro átomo é uma palavra que pertence a uma das seguintes classes gramaticais: substantivos comum, adjectivos, verbo ou nome próprio;
- NE NUM - candidato a nome de entidade que não tem entrada na Wikipédia e, é composto por um valor numérico;
- NE ASPAS - candidato a nome de entidade delimitado por aspas (“”);
- NE DATA - candidato a nome de entidade marcado como data;
- NE HORA - candidato a nome de entidade marcado como hora;
- NAO NE - candidato a nome de entidade que não reúne nenhuma das características anteriores.

As interpretações de frases que têm análise sintáctica, ou seja, uma ou mais estruturas sintácticas, devem ser valorizadas face às que não têm. A pontuação atribuída a essas interpretações, é um passo importante na escolha da melhor interpretação.

As interpretações que têm uma ou mais estruturas sintácticas foram divididas em:

- TOTAL REP - interpretação totalmente representável por uma estrutura sintáctica;
- PARCIAL REP - interpretação parcialmente representável por uma estrutura sintáctica;

O REMUE recebe um conjunto de interpretações de cada frase, pontuando os candidatos a nomes de entidades de cada interpretação e a interpretação em termos de análise sintáctica.

Para a análise sintáctica desenvolveu-se um analisador sintáctico, recorrendo às gramáticas de cláusulas definidas (Definite Clause Grammars - DCGs).

Este analisador sintáctico verifica se as interpretações das frases de um texto são representadas por estruturas sintácticas e infere essas estruturas.

Na construção da estrutura, por cada palavra da frase a analisar são verificadas as suas características no dicionário local.

2.4 Saída

A saída de um ficheiro processado pelo REMUE é constituída pela interpretação mais correcta de cada frase, ou seja, a interpretação que recebeu maior pontuação da função heurística.

Na saída, são gerados dois ficheiro, um com o texto em que os nomes de entidades são destacados das restantes átomos com etiquetas e um outro ficheiro que contém por linha: o número da frase, a lista de nomes de entidades e o valor de heurística atribuído.

3 Avaliação

Na avaliação do REMUE utilizaram-se 3 métricas que são usadas na avaliação de sistema de recolha de informação: precisão, cobertura e medida F. Estas métricas foram adaptadas ao problema de marcação de nomes de entidades.

Na avaliação comparam-se os ficheiros com a lista de nomes de entidades marcados manualmente e a lista de nomes de entidades marcados pelo REMUE.

O resultado da avaliação é escrito, frase a frase, num ficheiro com as seguintes variáveis: número da frase, número de nomes de entidades marcados, número de nomes de entidades correctos dos marcados, número de nomes de entidades incorrectos dos marcados, número de nomes de entidades existentes, precisão, cobertura, medida F e valor da heurística.

Na avaliação do REMUE foram utilizados 3 corpura: 700 perguntas do CLEF, 300 notícias do jornal o Público e 200 frases do Harem.

O CLEF (Cross-Language Evaluation Forun) é uma série de avaliações conjuntas que promove a pesquisa e desenvolvimento na área de recolha de informação entre várias línguas. A participação do português tem sido financiada pela Linguateca, a nível de recursos humanos, e pelo diário Público (Portugal) e Folhas de São Paulo (Brasil), a nível de fornecimento de recursos. A Linguateca disponibiliza a colecção CHAVE que contém textos, tópicos e perguntas utilizados nas edições do CLEF.

O corpus do CLEF foi usado para verificar até que ponto o uso da gramática desenvolvida para frases interrogativas tem impacto no desempenho do REMUE neste corpus.

O corpus do CLEF foi anotado manualmente identificando os nomes de entidades das frases, ou seja, sem classificação.

O corpus do Público foi utilizado para testar o desempenho do REMUE em frases para as quais a gramática tem um mau resultado, não consegue obter análise sintáctica em mais de 70% das frases do corpus.

O corpus do Segundo Harem foi utilizado para obter uma avaliação independente, ou seja, com as anotações feitas pela equipe da Linguateca. Apesar de usarmos as marcações do Harem, a avaliação foi feita por nós e só para uma amostra da Colecção Dourada do Segundo Harem, o que nos impede de comparar os nossos resultados com os dos sistemas que concorreram ao Segundo Harem. Fizemos os testes sobre uma amostra do Segundo Harem porque a nossa gramática que foi construída para frases interrogativas tem um mau desempenho (tempo e espaço) na análise das frases deste corpus.

Com cada um dos conjuntos de textos foram realizados 8 testes, nos quais foram feitas variações das pontuações atribuídas aos grupos de candidatos a nomes e às interpretações de frases representáveis por estruturas sintácticas.

Com estes testes procuramos estudar o impacto da informação: morfo-sintáctica, semântica e sintáctica no desempenho do sistema.

3.1 Heurística Utilizada: Pontuações Atribuídas

Em baixo pode ver-se a heurística usada no Teste 1. Os valores da heurística correspondem aos que dão o melhor desempenho ao REMUE nos diferentes corpura. Estes valores foram encontrados por tentativa e erro e não garantimos que sejam os melhores.

- NE WIKI $= 10 * (1 + Es)$ (Es -número de átomos).
  Neste teste valorizamos o facto de os nomes de entidades terem entrada na Wikipédia e também preferimos os nomes de entidades mais compridos (com mais átomos) se existirem na Wikipédia.
- NE SIMPLES $= 4$ e NE COMPOSTO $= 9 * (1 + Es)$ (Es -número de átomos)
  Valorizamos o comprimento dos nomes de entidades mesmo quando não temos entrada na Wikipédia.
- NE NUM$= -9$
  Desvalorizamos números isolados que sejam marcados como nomes de entidades.
- NE ASPAS $= 15$, NE DATA $= 15$ e NE HORA $= 15$
  Valorizamos expressões entre aspas que sejam marcadas como nomes de entidades e outras expressões que sejam marcadas como data e hora.
- NAO NE $= -10 * (1 + Es)$ (Es -é número de espaços em branco numa cadeia de caracteres)
  Desvalorizamos expressões que não satisfazem os nomes de entidades anteriores.
- TOTAL REP $= 100$ e PARCIAL REP $= 60$
  Com estes valores, valorizamos bastante o facto de a frase com os nomes das entidades marcados ter análise sintáctica (TOTAL REP=100 e PARCIAL REP=60)

Na Tabela abaixo podem ver-se os parâmetros usados na definição da heuristíca para cada teste.

	Heuristica
NE WIKI	NE1*(1+Es)
NE SIMPLES	NE2
NE COMPOSTO	NE3*(1+Es)
NE NUM	NE4
NE ASPAS	NE5
NE DATA	NE6
NE HORA	NE7
NAO NE	NE8*(1+Es)
TOTAL REP	I1
PARCIAL REP	I2

	Teste1	Teste2	Teste3	Teste4	Teste5	Teste6	Teste7	Teste8
I1	100	0	100	100	100	100	100	100
I2	60	0	60	60	60	60	60	60
NP1	10	10	0	10	10	10	10	10
NP2	4	4	4	5	4	4	4	4
NP3	9	9	9	5	9	9	9	9
NP4	-9	-9	-9	-9	0	-9	-9	-9
NP5	15	15	15	15	15	0	15	15
NP7	15	15	15	15	15	15	0	15
NP8	15	15	15	15	15	15	15	0
Es	N	N	N	0	N	N	N	N

Com estes testes pretendemos ver o impacto:

- teste 2 – da análise sintáctica. Não se tem em conta a existência de estrutura sintáctica para a frase com os nomes de entidades.
- teste 3 – da Wikipédia. Não se tem em conta a informação sobre a existência de entrada para o nome da entidade na Wikipédia.
- teste 4 – do comprimento do nome da entidade. Não se valoriza o número de átomos do nome da entidade.
- teste 5 – dos números isolados. Não se valorizam os números isolados que sejam marcados como nomes de entidades.
- teste 6 – das expressões entre aspas. Não se valorizam as expressões entre aspas que sejam marcadas como nomes de entidades.
- teste 7 – das datas. Não se valorizam as expressões que sejam marcadas como datas.
- teste 8 – das horas. Não se valorizam as expressões que sejam marcadas como horas.

	Prec	Cob	Med-F
Teste 1	0,9836	0,9804	0,9808
Teste 2	0,9765	0,9746	0,9727
Teste 3	0,9315	0,9335	0,9313
Teste 4	0,8949	0,9029	0,8975
Teste 5	0,9779	0,9796	0,9780
Teste 6	0,9746	0,9768	0,9748
Teste 7	0,9779	0,9796	0,9780
Teste 8	0,9779	0,9796	0,9780

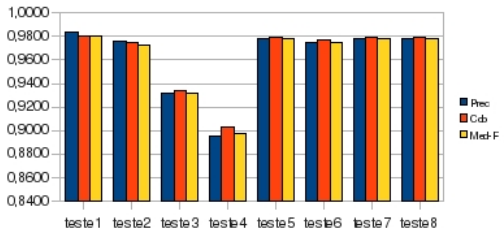


Figura 2. Testes com o corpus do CLEF

4 Conclusões e Trabalho Futuro

Na avaliação do REMUE foram utilizados 3 corpura: 700 perguntas do CLEF, 300 notícias do jornal o Público e 200 frases do Harem. Com cada um destes corpura foram realizados 8 testes com variações de alguns dos parâmetros que pontuam os nomes de entidades marcados e as interpretações de frases com análise sintáctica.

Como se pode ver na secção 3, onde se apresentam os resultados da avaliação do REMEU para 3 domínios diferentes de frases, o teste 1 foi aquele que apresentou os

	Prec	Cob	Med-F
Teste 1	0,9461	0,9547	0,9457
Teste 2	0,9450	0,9547	0,9450
Teste 3	0,8931	0,9238	0,8989
Teste 4	0,6945	0,7750	0,7210
Teste 5	0,8743	0,9547	0,8975
Teste 6	0,9003	0,9259	0,9042
Teste 7	0,9304	0,9547	0,9337
Teste 8	0,9304	0,9547	0,9337

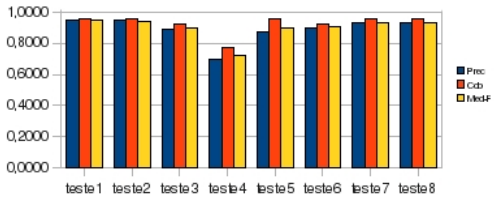


Figura 3. Testes com corpus do Público

	Prec	Cob	Med-F
Teste 1	0,7659	0,8877	0,8223
Teste 2	0,7559	0,8877	0,8165
Teste 3	0,6684	0,7194	0,6929
Teste 4	0,7126	0,8397	0,7710
Teste 5	0,7630	0,8827	0,8185
Teste 6	0,7582	0,8757	0,8127
Teste 7	0,7659	0,8827	0,8202
Teste 8	0,7659	0,8827	0,8202

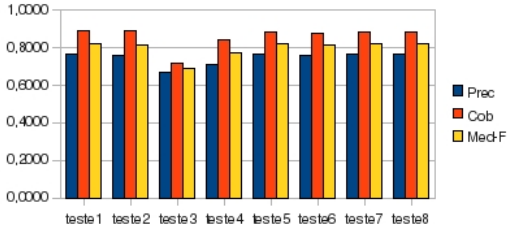


Figura 4. Testes com corpus do HAREM

melhores resultados nos 3 corpura. Neste teste valorizamos os nomes de entidades que se encontram na Wikipédia, as expressões entre aspas, as datas e as horas. Além disso, também valorizamos as interpretações de frases que contêm estruturas sintácticas.

No teste 2 não temos em contam se com os nomes de entidades marcados, a frase tem análise sintáctica. Só nos corpura do CLEF e Harem se vê diferença, 0.8% e 1.0%, entre o teste 1 e o teste 2, no corpus do Público a diferença é quase nula. Como a gramática utilizada não tem um bom desempenho nas frases do Público e do Harem, vamos procurar repetir os testes com uma gramática que tenha um melhor desempenho para estes textos. No entanto podemos concluir que o uso de uma gramática pode melhorar o desempenho do REMUE, ainda que de forma ligeira. Também podemos concluir que o uso da gramática nunca baixa o desempenho do REMUE.

O teste 3 também demonstra que nos 3 corpura os resultados baixam, 16% na cobertura do Harem, 3% na cobertura do Público e 4% na cobertura do CLEF. Permite-nos concluir que o uso de uma enciclopédia melhora os resultados.

O teste 4 retira o peso ao comprimento dos nomes de entidades, os resultados nos diferentes corpura permite concluir que escolher os nomes mais compridos é uma boa estratégia.

Os outros testes mostram que esta informação não tem grande impacto no desempenho do REMUE.

O REMUE pode evoluir em diferentes direcções mas as mais imediatas incluem: utilizar uma gramática com maior cobertura, transportar o REMUE para o inglês, e incluir a classificação das entidades mencionadas.

Referências

[AA08] Marcelo Adriano Amancio and Sandra Maria Aluísio. Explicitação de entidades mencionadas visando o aumento da inteligibilidade de textos em português. Technical report, Universidade de São Paulo, Agosto de 2008.

[AFM+08] Carlos Amaral, Helena Figueira, Afonso Mendes, Pedro Mendes, Cláudia Pinto, and Tiago Veiga. Adaptação do sistema de reconhecimento de entidades mencionadas da priberam ao harem. In Cristina Mota and Diana Santos, editors, *Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: O Segundo HAREM*. Linguateca, 2008.

[NS07] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 1(30):3–26, 2007.

[QRPV06] Paulo Quaresma, Irene Rodrigues, C. A. Prolo, and R. Viera. Um sistema de pergunta-resposta para uma base de documentos. *Letra de Hoje - Revista da Pontifíca Universidade Católica do Rio Grande do Sul*, 41(2):43–63, Junho 2006.

[SC07] Diana Santos and Nuno Cardoso, editors. *Reconhecimento de entidades mencionadas em português*. Linguateca, 2007.

[SR04] Diana Santos and Paulo Rocha. Chave: topics and questions on the portuguese participation in clef. In C. Peters and F. Borri, editors, *Cross Language Evaluation Forum: Working Notes for the CLEF 2004 Workshop*, pages 639–648, Bath, UK, September 2004 2004.

RuDriCo2 - a faster disambiguator and segmentation modifier

Cláudio Diniz, Nuno Mamede, João D. Pereira

IST – Instituto Superior Técnico
L^2F – Spoken Language Systems Laboratory – INESC ID Lisboa
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{Cdiniz,Nuno.Mamede,Joao}@inesc-id.pt

Abstract. Currently, L^2F's NLP chain has a bottleneck. Module RuDriCo (Rule Driven Converter) is substantially slower than the remaining modules of the chain. RuDriCo is a rule-based morphological disambiguator with the possibility to change segmentation (join or split tokens). This paper describes the changes made to the system to improve its performance by using the concept of layers and also by reducing the number of variables contained in the rules. It also describes the changes in rule syntax, such as the addition of new operators and contexts, which makes the rules more expressive.

Resumo. Actualmente, a cadeia de PLN do L^2F tem um módulo que é substancialmente mais lento que os outros, o RuDriCo. O RuDriCo é um desambiguador morfológico baseado em regras que também permite alterar a segmentação de texto. Este trabalho descreve os melhoramentos realizados, nomeadamente a introdução de novos operadores, a introdução do conceito de camada e a redução do número de variáveis usadas na especificação das regras.

1 Introduction

Natural Language Processing (NLP) is one of the most important Artificial Intelligence research areas. Many of the systems developed in this area, such as dialog systems or spelling correction systems, use a set of modules responsible for processing text. Usually such systems are organized in a pipeline and are referred to as *NLP chain*. Currently, the L^2F[1] research group uses a NLP chain (see Figure 1) to identify and classify Named Entities, extract semantic relations between those entities, to mention only a few. RuDriCo is one of the modules of the L^2F NLP chain.

The L^2F NLP chain is organized as follows. The first module receives the text to process and tokenizes it, defining the segments that compose the text. Palavroso [Medeiros, 1995] is a morphological tagger that receives the result of this segmentation as input and associates all possible part-of-speech (POS) tags

[1] Spoken Language Systems Laboratory of INESC-ID Lisboa.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 573–584

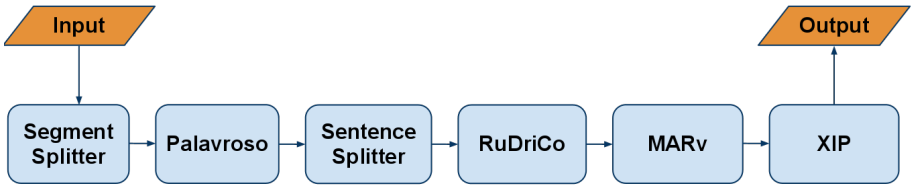


Fig. 1. L^{2}F's NLP chain

to each segment. The next module groups the segments into sentences. The next module to apply is RuDriCo [Pardal, 2007]. This module is a rule-based morphological disambiguator and it also makes segmentation changes to the input, like joining segments (compound words). MARv [Ribeiro et al., 2003], a stochastic morphological disambiguator, receives the result of RuDriCo and it selects the best POS tag to each segment. Finally, the last module to apply is XIP [Xerox, 2003] which is responsible for the syntactic analysis.

Disambiguation systems based on rules, also known as systems with linguistic knowledge [Màrquez and Padró, 1997], are the target of this study. The rules used in these systems are written by linguists. The rules consider the context of each word, and depending on the context make their disambiguation. This kind of methodology leaves some ambiguities unresolved, but is still common that current systems have an accuracy rate around 99%[2].

The input of RuDriCo is a set of rules and the text to process. Input text is in XML format and consists in a set of sentences where each sentence has one or more segments. The segments represent words that are constituted by a surface (`word`) and one or more annotations (`class`). An annotation is composed by a lemma (`root`) and a set of attribute-value pairs. The attribute-value pairs represent the properties of each annotation, e.g. the category of a word. For example, Figure 2 represents an ambiguous segment containing the word *partido*: it has one surface and three annotations.

RuDriCo has two types of rules: disambiguation and segmentation rules. The former ones allow the system to choose the correct category of a word by considering the surrounding context. Segmentation rules change the segmentation and can be divided into contraction and expansion rules. Contraction rules convert two or more segments into a single one. Expansion rules transform a segment into at least two segments. An example of an expansion rule is to transform the segment "Na" into two segments "Em" and "a". An example of a contraction rule is to turn segments "Coreia", "do" and "Sul" into a single segment "Coreia do Sul".

In the original RuDriCo, all types of rules share the same syntax:

```
antecedent --> consequent .
```

where the `antecedent` defines the conditions that must exist to perform the action specified in the `consequent`. In other words, RuDriCo tries to pair the

[2] The hit rate does not take into account the words that are not disambiguated.

```
<sentence>
  <word name="partido">
   <class root="partido">
    <id atrib="CAT" value="nou"/>
    <id atrib="SCT" value="com"/>
    <id atrib="NUM" value="s"/>
    <id atrib="GEN" value="m"/>
    </class>
   <class root="partido">
    <id atrib="CAT" value="adj"/>
    <id atrib="NUM" value="s"/>
    <id atrib="GEN" value="m"/>
   </class>
   <class root="partir">
    <id atrib="CAT" value="ver"/>
    <id atrib="MOD" value="par"/>
    <id atrib="NUM" value="s"/>
    <id atrib="GEN" value="m"/>
   </class>
  </word>
</sentence>
```

Fig. 2. The word "partido" represented in XML

`antecedent` with a sequence of segments from the XML input, and when it succeeds, that sequence of segments is replaced by the segments described in the `consequent`. The segment syntax is as follows:

```
'surface' ['lemma', 'prop_1'/'value1', 'prop_2'/'value2' ... ]
```

where `surface` and `lemma` are obligatory.

Figure 3 contains an example of a contraction rule that transforms the segments "Coreia", "do", and "Sul", in one segment with a single annotation.

```
'coreia' [L1,'CAT'/C1]
'do' [L2,'CAT'/C2]
'sul' [L3,'CAT'/C3]
-->
'Coreia do Sul' ['Coreia do Sul',
'CAT'/'nou','GEN'/'f','NUM'/'s'].
```

Fig. 3. Rule to join segments "Coreia", "do" and "Sul"

The RuDriCo main algorithm (see Figure 4) processes each sentence, segment by segment. A sentence is declared processed when the algorithm cannot apply any rule to it. When a rule is applied to a set of segments, the sentence is processed again to see if there is a new rule that can be applied. The Agenda algorithm (step 4) applies rules to input segments and is not explained since it falls out of the scope of this paper.

RuDriCo has 3 main disadvantages: (i) it is not sufficiently expressive: it does not have neither the "not" nor the "or" operator; (ii) it is not sufficiently

```
1: FOR EACH sentence S in text DO
2:    FOR each segment I in S DO
3:       agenda(I)
4:       IF (agenda(I) has applied a rule) THEN
5:          I = first segment of S
6:          GOTO 3:    /*first segment*/
7:       ELSE
8:          GOTO 2:    /*next segment*/
9:    ENDFOR
10: ENDFOR
```

Fig. 4. RuDriCo simplified algorithm.

efficient: it is the slowest module of the NLP chain; and (iii) it enters in infinite recursion whenever the antecedent of one rule matches the consequent of another rule, and the consequent of the first rule matches the antecedent of the second rule.

The goal of this work is to develop RuDriCo2, a faster RuDriCo with a more expressive and user-friendly syntax, and that avoids infinite recursion. This paper describes the improvements introduced in RuDriCo to implement RuDriCo2.

2 State of the Art

The morphological disambiguators can be classified according to the methodology that is used to solve the problem. [Cole et al., 1995] classifies these systems in two types:

- based on rules, where the knowledge (rules) is manually coded;
- stochastic, where the knowledge is automatically extracted from a previously manually annotated corpora.

Other authors classify these systems differently. For example, [Schmid, 1994b], [Schmid, 1994a] and [Schulze et al., 1994] classify these type of systems using different categories, based on neural networks. In this document, we will just consider Cole's classification. L^2F's NLP chain uses both types of morphological disambiguators: RuDriCo is based on rules and MARv is stochastic. Some of the most well known-rule based systems are:

- Computational Grammar Coder (CGC) [Klein and Simmons, 1963],
- TAGGIT [Greene and Rubin, 1962],
- EngCG [Voutilainen, 1995b] [Voutilainen, 1995a],
- Brill Tagger [Brill, 1992],
- XIP [Xerox, 2003],
- RuDriCo [Pardal, 2007].

CGC is a morphological analyzer and a disambiguator. It begins by addressing some exceptions which the morphological analyzer cannot deal, with a lexicon of 1500 words. After the morphological analyzer, the rule-based disambiguation

starts with about 500 rules. TAGGIT is based on the CGC and uses a larger vocabulary.

EngCG is not only a disambiguator, but it also performs some extra tasks such as the segmentation of the input text. The task sequence is the following: (i) segmentation; (ii) morphological analysis; (iii) morphological disambiguation; (iv) find other syntax tags; and (v) finite-state syntactic disambiguation. The morphological disambiguation task is seen as a set of rules. Each rule specifies one or more contexts where a label is false. A tag will be removed if a pattern is established. If a word has a single tag, the word is not ambiguous. This system leaves 3-7% of ambiguous words but their accuracy rate is 99.7%.

The system described in [Brill, 1992] is a morphological analyzer, but when it assigns tags to words, the context is analyzed. This system uses automatically learned rules to associate tags with the input text words. One of the drawbacks of rule-based systems is the need of human experts and linguists for the complex and time-consuming task of writing rules, but [Brill, 1992] shows that this effort can be reduced. The system begins by assigning the most likely tag to each word ignoring the context. Then it performs the learning task, which considers eight types of predefined rules. The system instantiates them and chooses the rules that have a lower error rate. After the rules are chosen, they are applied to the text. The author claims that this system can get better results if some rules are manually written.

The last system here considered is the XIP system [Xerox, 2003], which includes modules to perform morphological disambiguation, syntactic analysis and changes to the hierarchy of segments. Section 2.1 compares XIP and RuDriCo.

2.1 Comparing RuDriCo and XIP

In RuDriCo, the input data is a list of segments, but in XIP, it is a hierarchy of nodes. XIP has disambiguation rules, but it does not have contraction or expansion rules. XIP chunking rules include the following two types: sequence rules and immediate dominance rules. The sequence rules do something similar to a contraction, grouping several nodes into a new node that is added to a tree hierarchy. The difference between immediate dominance and sequence rules is that immediate dominance rules can not represent any order between the antecedent nodes. XIP still has two other types of rules that are not mentioned here since they fall out of the scope of this paper.

Table 1 summarizes the features of XIP and RuDriCo. As it can be seen, XIP does not have contraction rules, having, however sequence rules that change the hierarchical structure instead of the segmentation.

The syntax of a disambiguation rule in XIP is the following:

$$1> \text{noun,verb} = |\text{det}|\ \text{noun}\ |\text{verb}|$$

The number at the beginning of a rule is the rule layer. The rules are applied according to the layers they belong, starting with the rules of the layer with the lowest number. The rules which do not have a layer are placed in the higher

Features	RuDriCo	XIP
Disambiguation rules	x	x
Contraction rules	x	
Expansion rules	x	
Chunking rules		x
or operator		x
not operator		x

Table 1. Features of RuDriCo and XIP systems

priority layer, the layer number zero. The rule antecedent (`noun,verb`) indicates that there must be a segment with two annotations, a noun and a verb. The two sections between pipes (`|det|` and `|verb|`) are the contexts of the rule, the left context and the right context. The contexts mean that the segment that matches with the antecedent has to have a `det` before and a `verb` after. The rule consequent is the part between the contexts (`noun`), and it indicates which category should be chosen from the antecedent. In this example, the word is disambiguated to noun. This rule can be written in RuDriCo's syntax as the rule in Figure 5.

```
S0[L0,'CAT'/'det']
S1[L11,'CAT'/'nou'][L12,'CAT'/'ver']
S2[L2,'CAT'/'ver']
 -->
S0*
S1[L11,'CAT'/'nou']+
S2* .
```

Fig. 5. Disambiguation rule

Comparing the syntax of both systems, one concludes that XIP's rules are much more compact than RuDriCo's rules. In RuDriCo the lemma and the surface are always present in each item, but in XIP the surface and the lemma can be omitted. Rules do not always need to use the lemma, nor a surface, as the rule presented above. In RuDriCo, the way to ignore the lemma and the surface is by using variables (`S0`, `S1`, `S2`, `L0`, `L11`, `L2`, and `L12` in this example). When a rule does not want to change a segment surface, a variable must be used in the antecedent and the `*` operator on the consequent. This is a disadvantage compared to XIP, because the use of variables requires more computation and the rules are more complex. In conclusion, the syntax of RuDriCo is more complex than the syntax of XIP, less compact and less expressive (RuDriCo does not have logical operators).

3 Layers

In RuDriCo, all rules are stored into a single file and are considered at the same time by the rule matching algorithm. The rules are tested in the order they appear in this file. As an example, consider that the rules are organized as follows: first the expansion rules, then the contraction rules and at the end of the file the disambiguation rules. Then, expansion rules have an higher priority than any other type of rule, because they are placed at the beginning of the rule file.

Instead of loading a file with all the rules, RuDriCo2 loads a file with the filenames of the files that contain the rules. The layers of the rules are relative to the file they belong to. All layers of the first file have priority over the layers of the following files, regardless of their numbers. The number that represents the layer is only used to sort layers on that file.

To support the concept of layers, the rule processing algorithm, presented in Figure 4, was extended with a new cycle that goes through all the layers. This new cycle was added between steps 1 and 2.

Although adding complexity to the algorithm, the agenda algorithm solely runs with the rules of one layer at a time. The performance of the RuDriCo algorithm improves when the gain in the Agenda algorithm is larger than the loss of having an additional cycle.

Layers can also be used to solve the problem of recursion between rules. If the rules that generate recursion are placed in distinct layers then the recursion is avoided.

4 Syntax

Because the syntax of RuDriCo is not expressive enough to express linguistic knowledge, RuDriCo2 has an expanded syntax. The syntax is based on RuDriCo's original syntax, and the changes were made incrementally. The changes to the syntax of RuDriCo are:

- node description (Section 4.1);
- contexts (Section 4.2);
- new operators (Section 4.3);

4.1 Node description

In RuDriCo, when an item is described in a rule, the surface and the lemma are always present, even when their values are irrelevant. For instance, the rule specified in Figure 5 uses three variables (`L0`, `L12` and `L2`) that are not used in the consequent. The use of variables for this purpose can be avoided if the lemma and the surface become explicit attribute-value pairs. So, the properties "lemma" and "surface" are introduced.

The second change in the syntax to describe the attribute-value pairs consists in the absence of quotes (′) around the property names. Figure 6 contains an

example of the rule presented in Figure 5 in the new syntax. The surface property can only be used once in each item and the lemma can only be used once in each annotation (an annotation is represented between brackets). Notice that the rules can be represented on a more compact way in the new version.

```
[surface=S0,CAT='det'],
[CAT='nou'][CAT='ver'],
[surface=S2,CAT='ver']
-->
S0*
[CAT='nou']+
S2* .
```

Fig. 6. Disambiguation rule with new syntax

4.2 Contexts

Many of disambiguation rules use variables to simulate contexts, such as the rule shown in Figure 6. To avoid the use of variables for this purpose and to simplify the writing of the rules, contexts (composed by items) were introduced in the antecedent:

$$| \text{ left context } | \text{ Item}_1 \text{ Item}_2 \ldots \text{Item}_N | \text{ right context } |$$

Figure 7 contains the rule presented in Figure 6 rewritten with the new syntax. The use of the contexts turns the rules less extensive since there is no need to use variables to simulate contexts.

```
|[CAT='det']|
[CAT='nou'][CAT='ver']
|[CAT='ver']|
-->
[CAT='nou']+.
```

Fig. 7. Disambiguation rule with contexts

4.3 New Operators

In RuDriCo the negation operator has not been implemented, although it can be simulated by rule replication (when the negation is applied to properties with a finite set of possible values). For example, when the left context of the rule presented in Figure 7 is a token with any category except determinant, in RuDriCo it is necessary to spell out as many rules as the number of categories, except the one related to the determinant category. In RuDriCo2, the negation

```
|[CAT=~'det']|
[CAT='nou'][CAT='ver']
|[CAT='ver']|
 -->
[CAT='nou']+.
```

Fig. 8. Example of operator negation

operator (∼) has been introduced and it allows for the specification of this type of condition with a single rule as shown in Figure 8.

The lack of a disjunction operator is a similar problem. If it is necessary to make a disjunction between two values of a property, two rules have to be written, one for each value. For instance, imagine that in the rule of Figure 7, it is desired that the left context is a determinant or a preposition. In RuDriCo, two almost identical rules had to be written (see Figure 9). But in RuDriCo2, with the disjunction operator (;), this situation can be solved with a single rule, as it is shown in Figure 10.

```
|[CAT='det']|
[CAT='nou'][CAT='ver']
|[CAT='ver']|
 -->
[CAT='nou']+.
```

```
|[CAT='pre']|
[CAT='nou'][CAT='ver']
|[CAT='ver']|
 -->
[CAT='nou']+.
```

Fig. 9. Two rules to make a disjunction

```
|[CAT='det';CAT='pre']|
[CAT='nou'][CAT='ver']
|[CAT='ver']|
 -->
[CAT='nou']+.
```

Fig. 10. Example of disjunction

5 Evaluation

The evaluation of the syntax may be subjective, but Figure 11 shows that the same rule can be written in a more compact way. In segmentation rules, the number of characters in the new RuDriCo is 16% smaller than in the original RuDriCo. In disambiguation rules, the number of characters is 76.1% smaller than on the original. The improvement is much higher in disambiguation rules since they use more contexts and the disjunction operator.

The performance can be measured running the original RuDriCo and RuDriCo2 with the same input (set of rules and text input files). The performance of XIP was not compared with RuDriCo2's because it is difficult to convert the rules from one system to the other, and because some of the rules are not convertible since both systems have different expressive sintaxes.

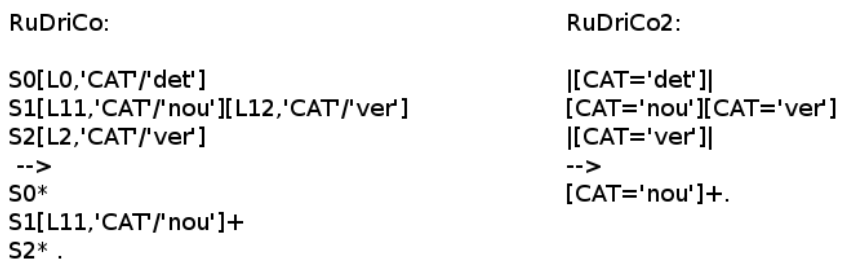


Fig. 11. Comparision between RuDriCo's and RuDriCo2's syntax

To test the performance, a set of 3096 rules was used with a set of text input files from CETEMPúblico[3], each one with a different size. The smallest file has only one sentence and the largest one has 50.000 sentences. Since changes were made incrementally, we have also evaluated an intermediate system, the RuDriCo with layers to assess the impact resulting from the introduction of layers.

The first performance evaluation aimed at discovering the optimal number of rules per layer. This study was done right after the implementation of layers. Since the disambiguation rules must remain on a single layer, only the other rules are used in this evaluation. The remaining rules (2330 rules) are divided into layers of equal size in order to find the optimal size of the layers. The tests were performed using an input text file with 1000 sentences and the results are shown in Table 2.

On one hand, when all rules (2330 rules) are kept in a single layer, which is the behavior of the original RuDriCo, the new RuDriCo spends 15.232 CPU seconds to process all the 1000 sentences. On the other hand, if there are 2330 layers, one rule per layer, then the system takes much longer to process them because the complexity of the algorithm for rule application depends on the number of layers. Results show that the use of layers improves the performance if each layer contains more than 32 rules. For the present set of rules and structure of the algorithm the optimal number of rules per layer is 167.

To evaluate the impact of the new syntax, the performance is measured comparing RuDriCo (the original), with RuDriCo with layers and the RuDriCo with

[3] Corpus with electronic extracts from the Público newspaper.

Rules/Layer	1	2	4	8	16	32	73	146	156	167	180	292	583	1165	2330
Time (s)	146	75.0	40.5	23.2	15.8	9.1	7.7	6.9	6.7	6.1	6.9	7.8	8.7	14.9	15.2

Table 2. Optimal number of rules per layer study

all features described in this paper, RuDriCo2[4]. The comparison is presented in Table 3.

The two smaller input files present only a small improvement because the system spends more time loading the rules than processing the input. For a text file with 1000 sentences, RuDriCo with layers needs 38,6% less time, and RuDriCo2 only needs 19.6% of the original time. To conclude, the new RuDriCo2 is about five times faster than RuDriCo in most cases.

Sentences per file	RuDriCo	RuDriCo with layers	RuDriCo2	% time of RuDriCo2 (comparing to RuDriCo)
1	0.15	0.19	0.11	73.3 %
10	0.20	0.74	0.18	91.8 %
100	8.33	3.36	1.69	20.0 %
500	38.00	15.37	7.83	20.1 %
1000	78.00	30.70	15.29	19.6 %
5000	392.00	152.19	76.80	19.6 %
10000	782.75	301.50	154.12	19.7 %
50000	can't process	1546.70	791.00	-

Table 3. Performance comparision

6 Future work and Conclusion

The experiments made show that RuDriCo2 is five times faster than RuDriCo. However, its efficiency can still be improved. The task that RuDriCo2 has to perform more times is the comparison between items of the rules and segments from the text input. This can be optimized using arrays of bits to represent the segments and the text input restrictions. So to test if an item pairs with a segment, the system will only have to perform a logic operation. The representation of items and segments in arrays of bits is the next scheduled improvement.

RuDriCo takes part of the L2F NLP chain, used to process text and it is the bottleneck. Besides this performance problem, RuDriCo does not support an expressive rule specification language. Is this paper we showed the changes performed in RuDriCo to address these issues. The layers and contexts make RuDriCo2 about five times faster than the original RuDriCo for the current set

[4] The original rules were automatically converted to the new syntax and the results are the same.

of rules. The addition of disjunction and negation operators makes the syntax of RuDriCo2 more expressive than the RuDriCo. The addition of contexts and the new node description allow RuDriCo2 rules to become more compact and easier to write. To conclude, RuDriCo2 is a significant improvement over the original module.

References

[Brill, 1992] Brill, E. (1992). A simple rule-based part of speech tagger. In *Proc. of the third conference on Applied natural language processing*, pages 152–155, Morristown, NJ, USA. Association for Computational Linguistics.

[Cole et al., 1995] Cole, R. A., Mariani, J., Uszkoreit, H., Zaenen, A., and Zue, V. (1995). Survey of the State of the Art in Human Language Technology, Center for Spoken Language Understanding CSLU, Carnegie Mellon University, Pittsburgh, PA.

[Greene and Rubin, 1962] Greene, B. B. and Rubin, G. M. (1962). Automatic Grammatical Tagging of English. Technical Report, Brown University, Providence, RI.

[Klein and Simmons, 1963] Klein, S. and Simmons, R. F. (1963). A Computational Approach to Grammatical Coding of English Words. In *Journal of the Association for Computational MAchinery (10)*, pages 334–347.

[Medeiros, 1995] Medeiros, J. C. (1995). Processamento Morfológico e Correcção Ortográfica do Português. Master's thesis, IST - Univ. Técnica de Lisboa, Portugal.

[Màrquez and Padró, 1997] Màrquez, L. and Padró, L. (1997). A Flexible POS Tagger Using an Automaticalluy Acquired Language Model. In *Proc. of the 35th Annual Metting of the Association for Computational Linguistics*, pages 238–245, Madrid.

[Pardal, 2007] Pardal, J. (2007). Manual do Utilizador do RuDriCo. Technical report, Instituto Superior Técnico - Universidade Técnica de Lisboa, Portugal.

[Ribeiro et al., 2003] Ribeiro, R., Mamede, N. J., and Trancoso, I. (2003). *Computational Proc. of the Portuguese Language: 6th Intern. Workshop, PROPOR 2003, Faro, Portugal, June 26-27, 2003*, volume 2721, Using Morphossyntactic Information in TTS Systems: Comparing Strategies for European Portuguese. Springer.

[Schmid, 1994a] Schmid, H. (1994a). Part-of-Speech Tagging with Neural Networks. In *Proc. of the 15th Inter. Conf. on Computational Linguistics*, Kyoto, Japão.

[Schmid, 1994b] Schmid, H. (1994b). Probabilistic Part-of-Speech Tagging using Decision Trees. In *Proceedings of the 15th International Conference on new methods in language processing*, Manchester, Reino Unido.

[Schulze et al., 1994] Schulze, B. M., Heid, U., Schmid, H., Schiller, A., Rooth, M., Grefenstette, G., Gaschler, J., Zaenen, A., and Teufel, S. (1994). Decide. MLAP-Project 93-19 D-1b I, STR and RXRC.

[Voutilainen, 1995a] Voutilainen, A. (1995a). *Constraint Grammar: a Language-Independent System for Parsing Unrestricted Text*, chapter Morphological Disambiguation. Mouton de Gruyter.

[Voutilainen, 1995b] Voutilainen, A. (1995b). A systax-based par-of-speech analyser. In *Proceedings of 7th Conference of the European Chapter of The Association for Computational Linguistics*, Dublin.

[Xerox, 2003] Xerox (2003). Xerox Incremental parser – Reference Guide.

Internet das Coisas e Serviços

Bridging the Browser and the Server

Miguel Raposo and José Delgado,

Instituto Superior Técnico, Universidade Técnica de Lisboa, Av. Prof. Cavaco Silva, Porto Salvo, Portugal
miguelfernandoraposo@gmail.com, jose.delgado@ist.utl.pt

Abstract. Web applications are now built on the principle that users interact with them through a generic, universal browser. The paradigm, client-server, is essentially limited to one-way interactions, with the client as the sole entity with real initiative. Also, server-based applications often do not guarantee information privacy, resulting in reluctance in its usage. This paper presents the Browserver as a means to give users the ability to be service providers, not mere consumers, and to avoid storing data at central servers. We describe an architectural approach and a technological solution for the union of a browser and a server for the development of a Browserver using existing technologies.

Keywords: Browser, Server, User Interface, Services, Peer-to-Peer

1 Introduction

In the early Internet days, applications were made with specific client and server side components (Fig. 1) and specific protocols, with interactions limited by the existence of the specific client on each user's machine. Nowadays, the browser constitutes a generic, universal client component capable of accessing all of the ever-growing Web applications (Fig. 2). Web users are seen as information generators, not merely as consumers. Although services already constitute the main paradigm at enterprise integration and the Internet of Services [17] is already a discussion subject, the Web is still centered around content and not on services, with the client-server paradigm limiting the interaction patterns with humans by requiring these to initiate the interaction by navigating to some page through a URL. If a user is involved in some business process, there is no direct way to interact with him through the browser so, the email is now the most used tool to contact and request someone's services, having become a nightmare and not practical for many persons nowadays.

To reduce this limitation, AJAX, polling and long-lived HTTP connections (Comet) [4] have been introduced to simulate server requests to the client, enabling more dynamic processes. Web Sockets [1, 2] are promising real bi-directional connections between the browser and the server, enabling better and faster communication between browser and server than AJAX. Nevertheless, the browser remains as a simple client, in the same paradigm, and business processes still depend on user's will to initiate the first interaction. This way, the email remains as an indispensable tool to connect people.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 587–598

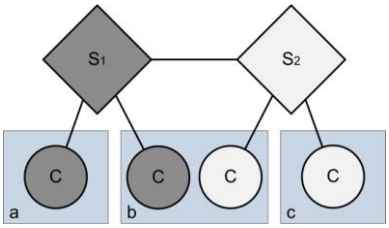


Fig. 1 A Specific client for each specific application.

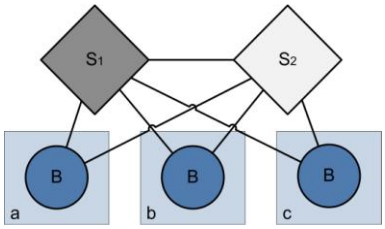


Fig. 2 The Web browser, a universal client for Web applications.

Using the browser, interactions between people in the Web always have a server as an intermediary, which offer the services that enable information sharing and collaboration. These intermediaries can take the information and use it for their own purposes. Users (or an agent on their behalf) should be able to provide electronic services themselves and be first class peers in web interactions and business or generic processes without the need for applicational intermediaries. We propose to overcome the limitations of the client-server paradigm by endowing each user not with a browser but with a Browserver (a browser (*B*) and a server (*S*)), as represented in Fig. 3. Interactions can be made directly between peers (*a*, *b*, *c*) equipped with a Browserver. Remote servers (*S1*, *S2*) can also be accessed as usual but are not as crucial. Direct, P2P interactions now become the norm instead of having to resort to centralized application servers for user interactions.

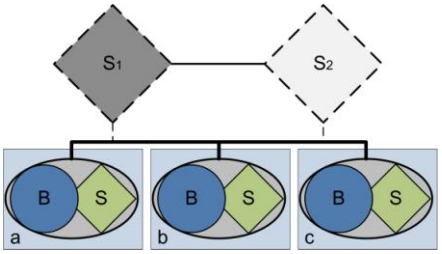


Fig. 3 The Browserver, the union of a universal client and a universal server on P2P interactions.

This entails a paradigm change for web usage, from client-server to peer-to-peer, and not just for file sharing. Applications such as email, instant messaging (IM), social networks, collaborative document edition and workflow systems can be implemented without necessarily depending on some central server system.

We conceptually present the Browserver in Sect. 2, a technological solution to it in Sect. 3, the related work in Sect. 4 and draw some conclusions in Sect. 5.

2 The Browserver

We consider a service as a capacity exhibited by an entity (e.g. a user or system) which can be offered by him, as provider and used by other(s), as consumer(s). The

Browserver provides a platform for service interactions involving users either as providers and/or consumers, including both:

- A Web browser. A generic and universal browser (e.g. Internet Explorer, Firefox, Chrome);
- A Web server. A generic and universal application server, enabling the user to provide services to other entities (e.g. Sun Glassfish, Apache Tomcat).

Although each user is able to create content and resources [13] that others can use, he is still positioned in the edge, and not in the center of the Web. Security issues limit browser's connections to be made only with the originating server of the Web page that the user is navigating, making impossible to build applications for direct collaboration through the browser. Each user acts as the ultimate consumer of services made available by other users or organizations on remote servers, that act as interaction intermediaries and have full access to information shared between peers even if private. The Browserver sets P2P (Fig. 5), as its paradigm for interactions, instead of the classic client-server model of the Web (Fig. 4).

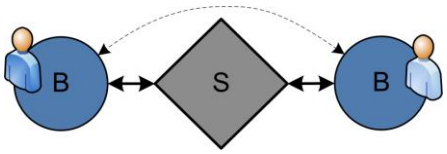


Fig. 4 Client-server model. There is always an intermediary offering services to each entity.

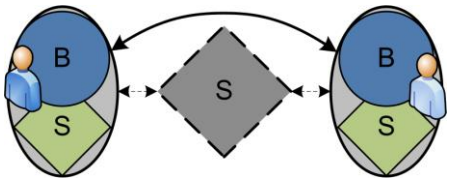


Fig. 5 Peer-to-peer model. Entities can interact directly, consuming each other's services.

The Browserver aims at giving each user a really active role in the Web, minimizing the need for intermediaries, and turning each user to be seen as an entity fully capable of providing services, rather than a mere consumer of information and services. The browser acts as a user interface for locally hosted services that can be made available to the Web as well as to remote services that need to interact with the user. Each public service of the user can be directly consumed (called, requested) by the entity who needs to. Everyone becomes a service provider and the Web becomes service centric instead of content centric.

To a business process, a person with a Browserver is seen as the set of invocable services that he provides. Also, services otherwise located at centralized servers may now exist in each user's computer. This entails:

- More information privacy, by putting services locally to each Browserver and directly consuming other's services in a peer-to-peer fashion;
- A complete service paradigm on building applications from which enterprise applications and customer relationship management can benefit from;
- New enterprise and personal relationships, and new tools for collaboration.
- Interactions with users can be proactive and not only reactive to user's actions.
- The email and other communication platforms became accessory and not mandatory for communications and interactions involving persons in the Web.
- Offline work, which can be granted by having the needed services and resources for an application executing at the local server.

3 A Technological Approach

In this section we present a high-level description of one solution for a technological implementation of the Browserver which is part of a work in progress on the subject. Although privacy requires security, that is not the focus of this article. The main focus on this architectural design goes to the connection of a browser and a server and automatic UI generation for services.

This approach intends to demonstrate the use of existing technologies to build a Browserver. Given that services are the paradigm of the Browserver, Web Services are chosen for its expressiveness and widespread use at organizational level and Java is chosen for its full support on the technology. However, the Browserver is not limited to a specific language or protocol.

3.1 Browser and Server

To unite a browser and a server some alternatives arise:

a) Develop from scratch a new fully integrated Web browser and server.
b) Develop a standards compliant browser frontend as an application running on the server.
c) Connect a local browser to a local server using existing solutions.

In the solution presented in this article, we opt for the last option. This gives the user the option to use the browser of its choice, while empowering him with the features of a Browserver. It also allows normal Web navigation, making the Browserver network a parallel Web to the existing one. Another advantage comes with the possibility of physical separation of both components. On user's will or necessity, its private server could be located remotely (at his home or office) and the browser could be on his mobile device (less computational capable).

The server must be compliant with Web Services [18] standards. Being Java the programming language, Java Servlets are used in the implementation, therefore Glassfish is the choice as it meets the requirements, with the integrated Metro web service stack [10]. Tomcat or other compliant server could have also been chosen.

To actively make requests to the user, Comet and Reverse-Ajax [4] help to overcome the limitations of the client-server model. Comet refers to long-lived HTTP connections, enabling low-latency communication between browser and server. Reverse-Ajax uses continuous *polling* from the client to the server for changes or server *pushing* to the client using Comet connections enabling a server to send data to the client without it without having been explicitly requested.

Direct Web Remoting [9], offers a framework for browser-server interaction based on Reverse-Ajax. Complementing with a strong Javascript library, like JQuery [16] full manipulation of a Web page displayed on the browser can be made. Fig. 6 shows:

- DWR Javascript library at the client side.
- DWR Java Servlet at the server side
- Browserver auxiliary and structural Javascript for UIs at the client side.
- Browserver Plain Old Java Classes (POJOs), Servlets and Beans, composing the Browserver architecture at server side.

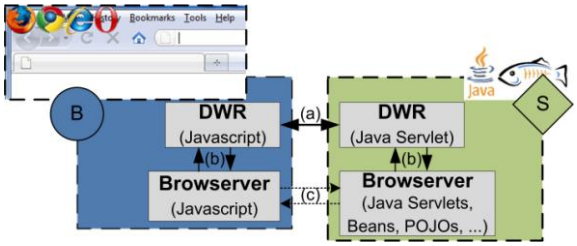


Fig. 6 Connecting browser (B) and server (S) through the DWR [9] framework.

The DWR framework exposes classes and methods on the Server that can be called from the client, being reverse Ajax used to connect both ends (*a*) through pull and push based techniques. At the client side, server methods are called (*c*) through the DWR framework (*b*), being the returned result obtained through a callback function.

The server also acts as a proxy to the browser, allowing navigation on the Internet. Fig. 7 shows a request (*a*) to a remote resource (a Web 2.0 site), going through a local proxy at the server that executes the request (*b*) and sends the response to the client. The response can be parsed, filtered and modified, if a service with such properties exists in the server, enabling the system to act as in [8].

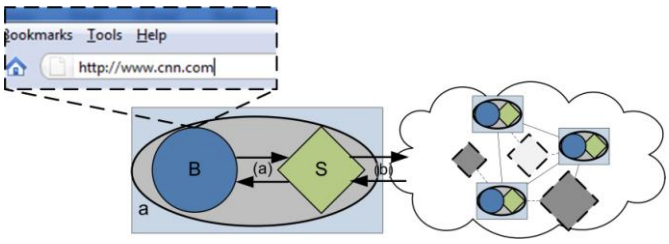


Fig. 7 The server (*S*) acting as a Web proxy to the browser (*B*).

3.2 Architectural Logical View

Fig. 8 presents a simplified logical view of the Browserver with two main parts: the *Browser (B)* and the *Server (S)*. In the context of this solution, the development leads to a single application deployed and running on the server. The *Browser* part of the system is responsible for creating and managing UIs for services and the connection with the browser. The *Server* part of the system has responsibility of managing services and the network of the Browserver. Each service has its own unique identifier, compliant with the URI syntax [11]. In the *Browser* part:

- The *BrowserManager*, coordinates the creation of *Containers* and *ContainerUnits*, and is responsible for sending the full container UI for the specific browser that requires it through the *Proxy*, as well as creating new UI units from *UIData* sent by the *ServiceManager,* using the *UnitBuilder*.
- A *UnitBuilder* takes the XML definition of an UI and builds a *ContainerUnit* representing that UI. The *BrowserManager* can then add it to a *Container*.

- There can be one or more *Container*s available and each holds multiple *ContainerUnits*, being *Portal*,*Portlets* and *ControlPanel* realizations of these. These elements produce code understandable by the browser like HTML and Javascript. The *Container* also updates the UI at the browser through the DWR whenever it changes internally.
- A *DataHandler* has the ability to handle user input from the browser. A *ContainerUnit* must handle this data, sending it to the *BrowserManager*, who forwards it to the *ServiceManager*.
- A Interface Unit can be:
  - A *SimpleUnit*, which cannot hold any other units inside (e.g. a *SimpleText* is used to present text without any special format).
  - A *ComplexUnit*, which can hold other units (e.g. in HTML, a <*div*> element plays this role).
  - A *DataUnit*, which is a *ComplexUnit* and *DataHandler* that collects data from the user (e.g. a <*form*> element in HTML corresponds to a DataUnit).
  - A *ContainerUnit*, which is a *DataUnit* that holds the whole UI for a service and handles input data from the browser, redirecting it to the corresponding service at the Server part.

Different browsers are supported by *Containers, ContainerUnits* and *UnitBuilders* that aim the specificities of each one. For a mobile device, a simple new *Container* that extends an existing one and converts the output using XSLT could be a solution.

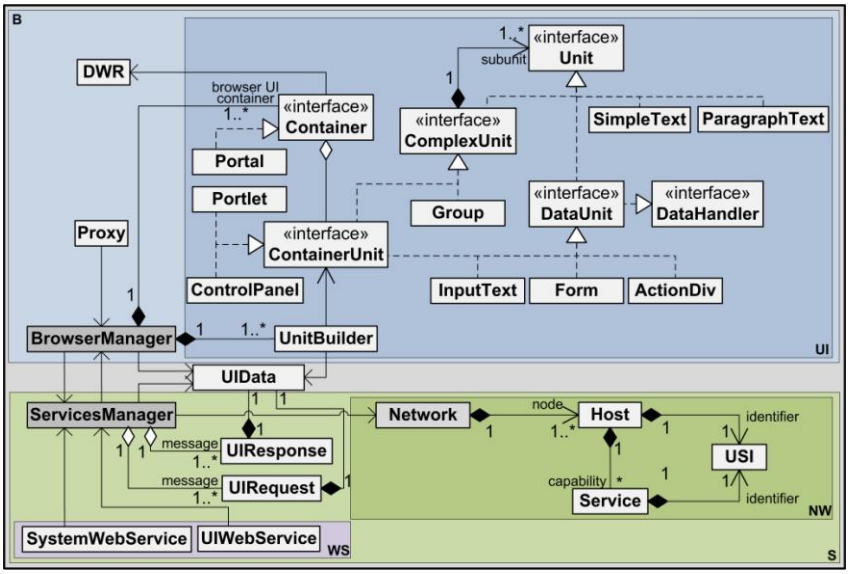


Fig. 8 Simplified logical view of the Browserver.

In the *Server* part:

- The *Server* part (S) is divided into the services part and the network (NW) part. In the services part, are the externally accessible Web Services (*WS*).

- The *ServicesManager* is responsible to manage incoming requests and outgoing responses for UI data.
- The *Network* consists of at least one *Host* (the local host) and all the known remote Browserver hosts that can have any number of associated *Service*s.
- The *Network* provides a means to remotely register local services. A remote service directory is used to publicize the services of the Browserver.
- The UIWebService is an externally accessible Web Service for external entities to request UIs to the local Browserver. *SystemWebService* is an externally accessible Web Service for external entities to make system requests. Its operations include *deployment* and *undeployment* of services. SOAP-based [22] implementations of these services offer great interoperability with existing systems.

3.3 Services

Services can be developed either to be local only or remotely accessible. The service is deployed on the local application server and registered in the Browserver, through the *SystemWebService*. The user has full control on whether the service is remotely published or not. Services can be composed of other services, promoting reusability.

Asynchronous communication is a crucial requirement on processes involving users so the Browserver *UIWebService* uses only one-way message exchange patterns. This decision is due to the nature of the behavior of users. A reply might be made immediately, or after weeks so, bidirectional communication channels can't be assured.

To receive replies to UI requests, the requester must provide a specific endpoint that is able to receive, process and correctly deliver SOAP messages, using WS-Addressing [22]. This is a limitation of existing communication channels, such as HTTP, which is the basis of the Browserver communication, as it is application-agnostic and can easily pass through firewalls.

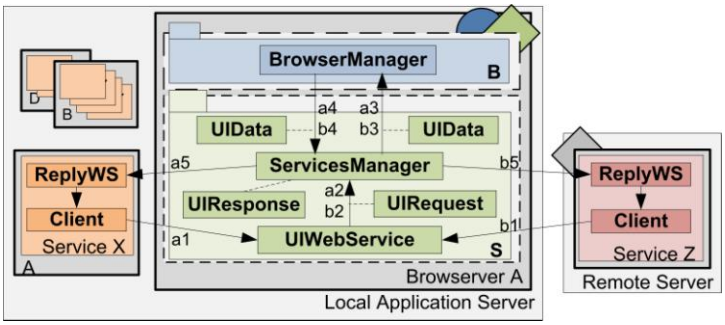


Fig. 9 Services activity on user interface request.

Fig. 9 presents a simplified activity on UI creation from services point of view:

1) A local service *X* and a remote service *Z*, request a UI to the Browserver, through the *UIWebService*, using SOAP messages (*a1,b1*) with UI definitions

compliant with the schema presented in Fig. 10. The messages include WS-Addresing headers indicating where to send the reply.

2) The *UIWebService* builds a *UIRequest* object with information provided by the sender and a *UIData* object representing the UI definition, sending it to the *ServiceManager* (*a2,b2*).
3) The *ServiceManager* dispatches requests, sending the *UIData* to the *BrowserManager*, that will generate and present the UI to the user (*a3, b3*).
4) The data submitted by the user is forwarded (*a4,b4*) by the *BrowserManager* to the *ServiceManager,* that builds a *UIResponse* with the data needed for the reply.
5) The *ServiceManager* sends the data (*a5,b5*) to the endpoint previously indicated by the requester.
6) The requester *Service* parses the data and act accordingly to its business rules.

To maintain context on successive service interactions, the *messageId* and *relatesTo* elements of the WS-Addressing headers are used. A user data response contains an ID that can be used on a later request, to indicate the relationship. To make a service publicly available, the user can indicate the Browserver to publish it in a service directory, like UDDI. A distributed solution for this is described in [21].

3.4 User Interface Generation and User Data Handling

Fig. 10 presents a simplified XML schema for UI definition for the Browserver. Upon receive a request, the *Browser* object uses the *UnitBuilder* to get a new *ContainerUnit* for that request. This *ContainerUnit* is then added to the *Container*, which has the responsibility to update the UI view at the browser, through DWR and JQuery.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:complexType name="interfaceType">
    <xs:group ref="iGroup" minOccurs="0" maxOccurs="unbounded"/>
    <xs:attribute name="title" type="xs:string"/>
    <xs:attribute name="id" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="formType">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="inputText" type="inputTextType"/>
        <xs:element name="text" type="xs:string"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="inputTextType">
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:group name="iGroup">
    <xs:sequence>
      <xs:choice>
        <xs:element name="p" type="xs:string"/>
        <xs:element name="text" type="xs:string"/>
        <xs:element name="form" type="formType"/>
        <xs:element name="group">
          <xs:complexType>
            <xs:group ref="iGroup" minOccurs="0" maxOccurs="unbounded"/>
```

```
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:sequence>
  </xs:group>
</xs:schema>
```

Fig. 10 Simplified user interface definition XML schema.

User input is gathered by Javascript (JQuery) and submitted to the *Browser* object through DWR as a JSON string object. No POST or other HTTP actions are activated at the browser. The JSON data is then converted to XML and sent to the *DataHandler* associated with the UI unit. The *DataHandler*, primarily the *ContainerUnit*, parses the data, deciding whether it will be redirected to a smaller unit to handle or to the requesting service as a data response message, whose schema is simplified in Fig. 11.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="data">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="input" type="dType" minOccurs="0"
maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="name" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Fig. 11 Simplified Data Response XML Schema.

3.5 Universal Service Identifier

The Universal Service Identifier (USI) is a Browserver approach that is not mandatory for the Browserver, as it is possible to implement the Browserver with other unique identifier scheme. We consider it to be a valuable means for unique service identification. A USI is a subset of the URI [11], with the following syntax:

urn:bs:[Browserver][Service][Operation] (1)

[Browserver] = [name]@[subdomain].[domain] (2)

The Service and Operation parts (1) are optional. When consisting only of the Browserver part (2), the it refers to the default Browserver service (the *UIWebService*). The schema of the Browserver services URNs can be exemplified:

- bs:mike@ist.pt/
- bs:mike85@ist.pt/id/Accounting
- bs:john@chunk.us/MathService/squareOp

The Universal Service Identifier has no existing implementations, so, compliant solutions would have to be developed. A solution comprising a distributed hierarchical architecture, like the DNS (that could eventually be adapted), providing services for naming and locating services, and Ad-UDDI [21] would fulfill the needs.

The USI offers a naming schema that can be used to fully resolve a service location, provided that the Web supports it with the necessary systems.

4 Related Work

There is currently no known work which is conceptually closely related to the Browserver, although there are some attempts to give the user the ability to provide services or resources using the browser.

Opera Unite [20] couples a browser and a server, giving the user the possibility of providing some services and resources to other users, but not in a direct fashion. Opera servers are always in the middle, and there is no continuous presence for services or resources in the Web.

In [19], the author tries to get the browser to be seen as web services server, but the method results in sending some notification (email, SMS) to the user with a URL to follow, instead of making a direct request to the browser which the user can fulfill.

Smart Browser [8], intends to provide more processing power, enabling background processing that can change the way things are presented, but doesn't give service provider capabilities to the user, who remains a mere consumer.

Most of current efforts to improve the Web are centered at the user experience as a consumer. The HTML5 draft enables more interactive content, by extending the dynamic UI creation to meet the standards. Anyhow, many capabilities it will bring to the browsers can be done by the local server and, for greater user interaction, also Flash can be used, so the choice isn't limited.

Still in a draft state of a standard protocol [2] and API [1], Web Sockets promise to enable seamless bi-directional communication and, consequently, much lower latency in connections between browser and server, even through intermediary proxies and firewalls (if encryption is used). The Browserver might eventually benefit from the use of such technology for communication, although the direction the technology is heading does not put the user in a provider position, as the services still remain at the servers.

5 Rationale and Conclusions

This work intends to be a first approach to the development of the Browserver, and instigate discussion over the best solutions to it as no system today implements its features. An implementation of the Browserver is under development as a demonstration of the concept, with the architectural design presented by this article mostly implemented and functional.

The Browserver is intended to be a platform for the Internet of Services and can change the way Web applications are designed. People, the leaves of the current Web, can be invoked as if they were Web Services. Workflows can be implemented by knowing that each participant is able to perform a task and to provide a service, directly requested (as in a real business process) and not relying on the user's willingness to follow an URL.

Nowadays, collaborative work is made mostly using central servers. Most companies prefer using their own infrastructure as a security and privacy measure. As in [15, 7], the Browserver eliminates the intermediaries in communication, therefore providing a platform for more secure and private collaboration environments. The e-mail is one of the applications that can be redesigned to send messages directly to the addressee or to feed them through some trusted third-party with user defined encryption mechanisms.

New and existing large-scale applications can be built or redesigned by knowing that the client has the ability to perform server-side tasks, lowering the load on the application servers. New P2P social networks are also a targeted application area. We can maintain a social network by keeping the URNs of all our connections, instead of having them all stored in some server.

Not all the current technologies are well suited for the Browserver. NAT constitutes an obstacle to P2P networks like the one the Browserver intends to build, and the existing solutions are not optimal. While Web Services are still the most used standard technology to implement the service paradigm, their complexity and sluggish performance constitute an opportunity for alternatives that best suit performance and scalability, such as WOA and REST [14]. However, expressiveness is not the strongest point in REST. The convergence of the two approaches is now the focus of study and development [17]. Peer-to-Peer networks using Web Services have already been addressed by [6, 3, 5, 12].

References

[1] The WebSocket API. W3C Working Draft, June 2010. http://dev.w3.org/html5/-websockets/, last access on 2010-07-14.

[2] The WebSocket protocol. IEFT Draft, May 2010. http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-00, last access on 2010-07-14.

[3] Conrad, M., Dinger, J., Hartenstein, H., Schöller, M., and Zitterbart, M.: Combining service-orientation and peer-to-peer networks. In KiVS Kurzbeiträge und Workshop, p. 181–184, 2005.

[4] Crane, D. and McCarthy, P.: Comet and Reverse Ajax: The Next-Generation Ajax 2.0. Apress, Berkely, CA, USA, 2008.

[5] Galatopoullos, D.G., Kalofonos, D.N., and Manolakos, E.S.: A P2P SOA enabling group collaboration through service composition. In ICPS '08: Proceedings of the 5th international conference on Pervasive services, pages 111–120, New York, NY, USA, 2008. ACM.

[6] Harrison, A., and Taylor, I.: Dynamic web service deployment using WSPeer. In Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies, pages 11–16. Louisiana State University, February 2005.

[7] Kortuem, G., Schneider, J., Preuitt, D., Thompson, T. G., Fickas, S., and Segall, Z.:When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. Peer-to-Peer Computing, IEEE International Conference on, 0:0075, 2001.

[8] Lin, D., Jin, J., and Xiong, Y.. Smart Browser: A framework for bringing intelligence into the browser. volume 7540, Tower A505 SP Tower, Tsinghua Science Park, HaiDian District, Beijing, China, 100084, 2010.
[9] Marginian, D. and Walke, J.: Direct Web Remoting - easy Ajax for Java, 2010. http://directwebremoting.org/, last access on 2010-06-07.
[10] Sun Microsystems. Metro, open source web service stack, 2009.
[11] T. Berners-Lee, Fielding, R., and Masinter, L.: RFC 3986, Uniform Resource Identifier (URI): Generic syntax. Request For Comments (RFC), 2005.
[12] Mondejar, R., Garcia, P., Pairot, C., and Skarmeta, A.F.G.: Enabling wide-area service oriented architecture through the p2pweb model. In WETICE '06: Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pages 89–94, Washington, DC, USA, 2006. IEEE Computer Society.
[13] Tim O'Reilly: What is web 2.0: Design patterns and business models for the next generation of software. MPRA Paper 4578, University Library of Munich, Germany, March 2007.
[14] Pautasso, C., Zimmermann, O., and Leymann, F.: Restful web services vs. "big"' web services: making the right architectural decision. In WWW '08: Proceeding of the 17th international conference on World Wide Web, pages 805–814, New York, NY, USA, 2008. ACM.
[15] Reif, G., Kirda, E., Gall, H., Picco, G.P, Cugola, G., and Fenkam, P.: A web-based peer-to-peer architecture for collaborative nomadic working. In 10th IEEE Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (Wetice), pages 334–339. IEEE Computer Society Press, 2001.
[16] John Resig. JQuery: The write less, do more, javascript library, 2010.
[17] Schroth, C. and Janner, T.: Web 2.0 and SOA: Converging concepts enabling the internet of services. IT Professional, 9:36–41, 2007.
[18] W3C: Web services architecture, February 2004. http://www.w3.org/TR/ws-arch/, last access on 2010-07-04.
[19] Waldorf, J.A., Lu, Y., and Demetriades, A.: Web browser as web service server in interaction with business process engine. Patent US 2005/0182768 A1, Aug 2005.
[20] Opera: Opera Unite. http://unite.opera.com/, last access on 2010-07-14.
[21] Du, Z., Huai, J., and Liu, Y. Ad-UDDI: An active and distributed service Registry. In C. Bussler and M.-C. Shan, editors, 6th VLDB Int'l Workshop on Technologies for E-Services, volume 3811 of LNCS, pages 58–71. Springer, 2006.
[22] Weerawarana, S., Curbera, F., Leymann, F., Storey, T, and Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

Execução de Fluxos de Trabalho com Simulação de Redes de Sensores

Duarte Vieira e Francisco Martins

Faculdade de Ciências da Universidade de Lisboa
LaSIGE & Departamento de Informática
Edifício C6 Piso 3, Campo Grande
1749 - 016 Lisboa, Portugal
dvieira@lasige.di.fc.ul.pt, fmartins@di.fc.ul.pt

Resumo As redes de sensores têm ganho relevância nas mais variadas áreas, com especial ênfase na monitorização ambiental e industrial e, mais recentemente, na logística. A informação recolhida do meio ambiente (pelas redes de sensores) pode influenciar o decurso dos fluxos de trabalhos destas áreas, pelo que, quando estes são representados em sistemas de gestão de fluxos de trabalho, testar o sistema como um todo pode tornar-se bastante complicado. Geralmente os testes efectuados nestes sistemas fazem uso de informação de registos de execuções de fluxos de trabalho anteriores. Alternativamente, os testes podem ser efectuados recorrendo a simulações de aplicações de redes de sensores. Para além de cobrir as situações descritas no caso anterior, esta abordagem permite testar novos fluxos, bem como testar variações introduzidas nos fluxos de trabalho por eventos do meio ambiente. Este artigo descreve uma forma de integrar plataformas já existentes com o objectivo de introduzir a simulação de redes de sensores nos testes de fluxos de trabalho.

1 Introdução

Uma rede de sensores sem fios é composta por uma colecção de dispositivos capazes de medir um determinado campo escalar ou vectorial e cuja comunicação entre os vários nós é feita sem fios. Numa rede de sensores podem existir nós com a capacidade de actuar sobre o ambiente (actuadores), bem como uma (ou mais) estação-base, responsável por processar os dados provenientes dos sensores e, quando necessário, (re)configurar o comportamento da rede.

As redes de sensores são um tópico que no passado recente cresceu em atenção quer por parte de empresas, quer por parte de grupos de investigação. Os desafios colocados ao nível do *hardware*, como a miniaturização dos dispositivos ou o aumento da capacidade da bateria e do alcance da comunicação sem fios, ombreiam com os desafios colocados ao nível do *software*, particularmente no que concerne a sistemas operativos e a linguagens de programação para estes dispositivos. Em ambas as áreas têm-se registado avanços importantes [3, 13, 20].

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 599–610

As aplicações das redes de sensores são inúmeras e abrangem, por exemplo, desde a leitura de sinais vitais do nosso organismo (*body sensor networks*), passando por redes de monitorização de condições ambientais (*e.g.*, medição da qualidade do ar ou dos oceanos, detecção de fogos florestais), até a aplicações espaciais [2]. Uma das áreas de aplicação mais recentes das redes de sensores é a *Internet das Coisas e Serviços* (IoT), que consiste em integrar o *estado do mundo* visto pelos *olhos* de sensores em aplicações de mais alto nível, disponibilizadas na *Internet*. Desta forma, as aplicações podem beneficiar de observações realizadas no ambiente onde estão integradas e adequar o seu comportamento de acordo com os valores lidos. Por exemplo, uma aplicação de *domótica* pode estender o âmbito das suas funcionalidades se, para além de oferecer o tradicional agendamento de tarefas (*e.g.*, ligar ou desligar um dado dispositivo, ou subir ou descer persianas de acordo com um plano pré-estabelecido), reagir em função de condições ambientais ou de acordo com regras comportamentais dos habitantes da casa. Outra área de aplicação é a logística, onde, por exemplo, se pretende adequar um processo de entrega às condições da mercadoria que está a ser transportada e às condições de trânsito no trajecto até ao destino. Neste caso, as informações obtidas através dos sensores podem originar uma alteração no processo de entrega, podendo resultar na alteração da ordem de entrega das mercadorias.

Este tipo de aplicações que descrevem fluxos de trabalho são difíceis de testar, pois baseiam o seu comportamento em acontecimentos externos (do mundo) que são, no caso geral, não deterministas. A abordagem mais comum para testar estas aplicações consiste em repetir a execução de fluxos de trabalho guardados. Embora simples, esta abordagem padece de diversas enfermidades, a saber: (a) só permite testar fluxos de trabalho que não sofreram alterações relativamente ao traço que está guardado; (b) não permite testar novos fluxos de trabalho definidos; e (c) não permite testar novas variantes de fluxos actuais. Outra abordagem ao teste destas aplicações consiste em obter informação do ambiente através da simulação das redes de sensores, permitindo a criação de ambientes virtuais onde se pode estudar o comportamento dos sensores *per se*, da rede como um todo, e da aplicação. Esta abordagem tem como desvantagem a construção de um modelo para simular redes de sensores. À medida que o âmbito de aplicação das redes de sensores se alarga é necessário recorrer a simuladores, que por si só constituem um desafio, pois simular uma rede de sensores com comportamentos não triviais está longe de ser uma tarefa simples e rápida. Além disso, a integração dos resultados destas simulações com o sistema de gestão de fluxos de trabalho constitui outro problema a mitigar.

Em [19] descrevemos um processo que permite criar automaticamente um modelo de simulação de uma rede de sensores a partir de especificações de alto nível. No presente artigo iremos focar na interacção da simulação de redes de sensores com uma ferramenta de execução de fluxos de trabalho. Na secção seguinte apresentamos um cenário na área da logística que ilustra a utilização de sistemas de gestão de fluxos de trabalho integrados com redes de sensores

para avaliar as condições em que se processa uma entrega de matérias sensíveis à temperatura.

A organização do artigo é a seguinte: a Secção 2 aborda a simulação de redes de sensores, exemplificando com uma linguagem de programação e com um simulador; a Secção 3 apresenta alguns sistemas de gestão de fluxos de trabalho e elabora na integração da simulação de redes de sensores com a execução de fluxos de trabalho; finalmente, na Secção 4 concluímos o artigo e indicamos algumas direcções para trabalho futuro.

2 Simulação de Redes de Sensores no Contexto dos Fluxos de Trabalho

A simulação de redes de sensores permite testar não só aspectos como a transmissão de sinais, ou a medição de grandezas físicas, mas também aspectos de mais alto nível, como protocolos de comunicação e execução de aplicações. A simulação tem, portanto, grande interesse em qualquer área de actividade em que as redes de sensores sejam utilizadas.

Consideremos o seguinte cenário que descreve um fluxo de trabalho integrado com recolha de condições ambientais, obtida por sensores: um camião parte de Lisboa com dois contentores de vacinas, cada um equipado com um sensor capaz de medir a temperatura. Em ambos a temperatura não deve exceder os 10°C, sob pena de colocar em risco a qualidade das vacinas. O primeiro contentor a ser entregue tem como destino Coimbra e o segundo o Porto. O camião está equipado com uma estação-base que, periodicamente, pede a medição da temperatura aos sensores presentes no camião e que informa o centro de controlo, via GSM, caso alguma das leituras ultrapasse os 10°C. No centro de controlo um fluxo de trabalho é iniciado quando é recebida comunicação. Testar este fluxo de trabalho obriga a que sejam simuladas comunicações do camião. Todavia, se a simulação da rede de sensores estiver integrada com o fluxo de trabalho, tal não será necessário. Mais, a integração pode permitir a utilização de dados de simulação de redes de sensores com um grau de complexidade maior que o do exemplo anterior.

Em [19] propusemos um gerador de modelos de simulação de aplicações para redes de sensores, obtendo resultados encorajadores, tanto em tempo de simulação como em utilização de memória, para aplicações que correm em várias centenas de sensores. Nesta secção, apresentamos a linguagem para programar redes de sensores Callas e o gerador de modelos de simulação.

2.1 Callas

Callas [14] é uma linguagem de programação que tem por objectivo estabelecer uma base formal para o desenvolvimento de linguagens e sistemas de execução para redes de sensores. Pode ser utilizada por si só para programar redes de sensores ou como linguagem intermédia sobre a qual se possam compilar outras linguagens, com maior grau de abstracção.

A linguagem Callas é *type-safe*. Esta propriedade garante que programas bem tipificados não produzem erros em tempo de execução, algo de extrema importância no contexto das redes de sensores, em que os testes e a depuração são difíceis ou mesmo impossíveis de se fazer após a sua instalação num ambiente real.

Uma aplicação Callas é composta por interfaces (ficheiros `.caltype`), um programa por tipo de sensor (ficheiros `.callas`) e uma descrição da rede (ficheiro `.calnet`). Apresentamos agora uma aplicação Callas para a rede descrita acima. A interface types.caltype define os dois tipos de módulos usados nesta rede: Nil, o módulo vazio, e AlertTemp, um módulo com as funções sample, que não tem parâmetros e que retorna Nil, e alert, com os parâmetros mac (endereço MAC, *string*), time (tempo, *long*) e temp (temperatura, *double*) e que também retorna Nil.

```
# file: types.caltype

defmodule Nil: pass

defmodule AlertTemp:
  Nil sample()
  Nil alert(string mac, long time, double temp)
```

O ficheiro iface.caltype define a interface da rede. Todos os nós na rede serão do tipo Sensor, que estende o tipo AlertTemp e acrescenta a função listen.

```
# file: iface.caltype
from types import *

defmodule Sensor(AlertTemp):
  Nil listen()
```

Embora toda a rede implemente a mesma interface (garantia verificada em tempo de compilação), os sensores podem ter comportamentos diferentes, consoante a implementação da interface. O ficheiro node.callas contém o programa para os nós da rede, ou seja, para os sensores dos contentores. O programa importa as interfaces de iface.caltype, define o módulo m, instala-o (**store** m) e escalona a execução periódica da função listen. O módulo m implementa (a) a função listen, que consiste somente no comando **receive**, que lê mensagens da fila de entrada, (b) a função sample, que envia para a rede a chamada a alert(mac, time, temp), com os valores lidos e (c) a função alert, vazia.

```
# file: node.callas
from iface import *

module m of Sensor:
  def listen(self):
    receive
```

```
  def sample(self):
    mac = extern macAddr()
    time = extern getTime()
    temp = extern getTemperature()
    send alert(mac, time, temp)

  def alert(self, mac, time, temp):
    pass

store m
listen() every 30000 expire 36000000
```

O programa em sink.callas implementa a mesma interface, mas com um comportamento diferente. Na estação-base a função sample apenas envia para a rede a chamada a sample(). A função alert regista os valores recebidos e, caso a temperatura seja superior a 10°C, transmite-os por GSM. Para além do escalonamento periódico à função listen, faz um outro, à função sample, que por sua vez envia para a rede o chamada a sample().

```
# file: sink.callas
from iface import *

module m of Sensor:
  def listen(self):
    receive

  def sample(self):
    send sample()

  def alert(self, mac, time, temp):
    extern logString(mac)
    extern logLong(time)
    extern logDouble(temp)
    extern logString(" ")
    if temp > 10.0:
      extern sendGSM(mac, time, temp)

store m
listen() every 30000 expire 36000000
sample() every 60000 expire 36000000
```

2.2 Simulação de Aplicações Callas

Podemos categorizar os simuladores de redes de sensores como *específicos*, caso em que o simulador modela apenas uma arquitectura/sensor, ou como *genéricos*,

caso em que o simulador permite a modelação dos próprios sensores. O VisualSense [5] é um simulador genérico e de código fonte aberto baseado na plataforma de modelação e simulação Ptolemy II [9], desenvolvida pela UC Berkeley. Permite (a) simular todos os aspectos das redes de sensores referidos anteriormente, (b) simular redes em que os nós podem correr código diferente entre si, uma característica invulgar nos simuladores em geral [8] e (c) modelar e simular em modo gráfico. No Ptolemy II a modelação é feita utilizando componentes (chamados *actores*, seguindo o Modelo de Computação por Actores [1]) que interagem apenas por troca de mensagens.

O gerador de modelos apresentado em [19] permite a parametrização do número e disposição dos nós da rede de sensores, da aplicação que corre e dos modelos dos sensores e de rede. A simulação dos modelos gerados obteve bons resultados em termos de desempenho e escalabilidade, conforme se pode verificar na Figura 1.

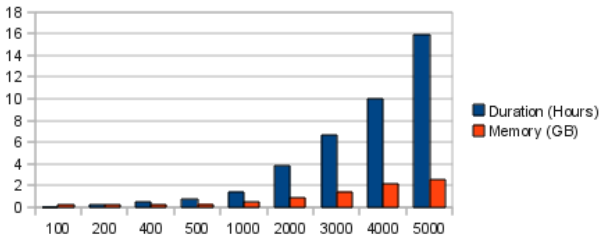


Figura 1. Duração da simulação e utilização da memória (abcissas) dado o número de sensores na rede (ordenadas).

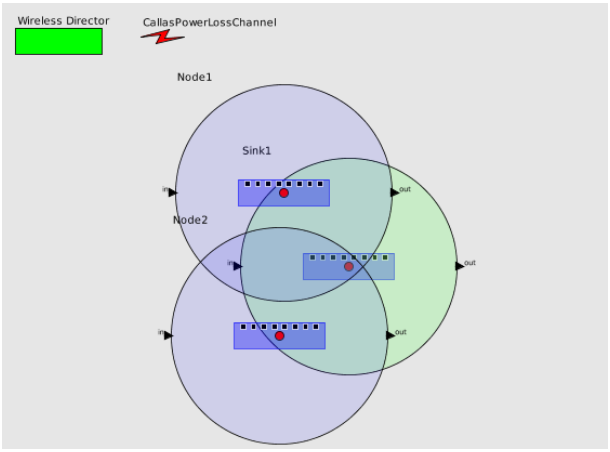


Figura 2. Rede de sensores no VisualSense.

O modelo de simulação para a aplicação definida na Secção 2.1 é gerado a partir do ficheiro network.calnet que, em adição à informação necessária à compilação do programa (interface a ser usada e o código de cada tipo de sensor), especifica, por exemplo, o número de nós e respectivas posições, o modelo do sensor (o formato persistente do Ptolemy II tem a extensão .moml) e o raio de comunicação. Com esta informação o gerador produz um modelo da rede que pode ainda ser editado no VisualSense, como se ilustra na Figura 2.

```
# file: network.calnet
interface = iface.caltype

sensor:
  code = node.callas
  size = 2
  range = 50
  position = random 0, 0 to 10, 10
  template = containerSensor.moml

sensor:
  code= sink.callas
  size = 1
  range = 50
  position = explicit 0, 0
  template = gsmSink.moml

template = containerNetwork.model # modelo do nível de rede
```

3 Execução de Fluxos de Trabalho em Kepler com Integração de Simulação de Redes de Sensores em VisualSense

Os sistemas de gestão de fluxos de trabalho, como o Sistema YAWL [18], são habitualmente utilizados para analisar processos de negócio, sendo também utilizados na validação dos processos de negócio em tempo de desenho [15].

Os sistemas de gestão de fluxos de trabalho científicos são uma especialização do tipo mais geral que permite executar fluxos de trabalho científicos (por exemplo, um conjunto estruturado de operações sobre dados provenientes de medições de grandezas físicas). Entre os sistemas deste tipo estão o Taverna [10], o Triana [16] e o Kepler [4].

O Kepler suporta modelação e execução em modo gráfico, composição de fluxos de trabalho, computação distribuída e acesso a repositórios de dados e serviços *web*. É um sistema baseado na plataforma Ptolemy II, tal como o VisualSense, mais propriamente, o Kepler e o VisualSense são especializações do Ptolemy II. A execução dos fluxos de trabalho é determinada pelo domínio de computação. Por exemplo, no domínio *Synchronous Dataflow* (SDF), a execução decorre de uma forma síncrona e numa sequência pré-calculada, enquanto que no

domínio *Process Networks* (PN) a execução é paralela, em que um ou mais componentes correm em simultâneo. Já o domínio *Discrete Event* (DE) é indicado para fluxos de trabalho com noção de tempo e eventos.

Uma vez que existe interoperabilidade de actores e directores (que implementam os domínios de computação) entre os dois sistemas, é possível integrar modelos de simulação de redes de sensores nos fluxos de trabalho definidos em Kepler de forma bastante directa. Deste modo, obtém-se uma forma de incluir informação proveniente da simulação de redes de sensores na simulação de processos de negócio. Adicionalmente, se tirarmos partido da linguagem Callas e do gerador de modelos de simulação, poderemos facilitar todo o trabalho relacionado com a simulação de rede de sensores.

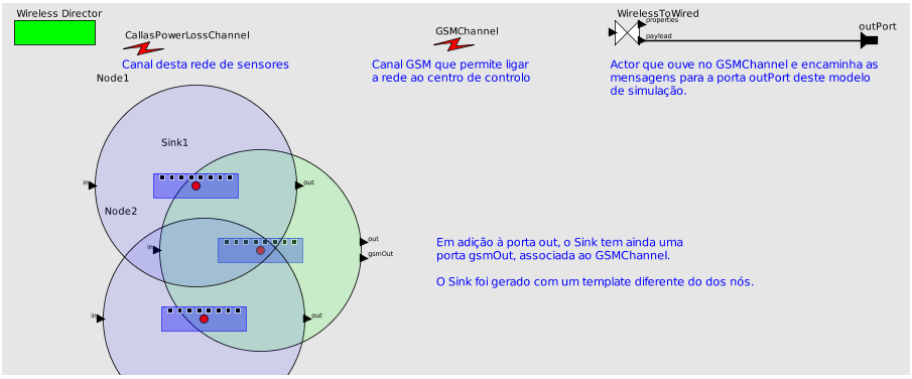


Figura 3. Modelo da rede de sensores para integração na execução do fluxo de trabalho

A Figura 3 apresenta o modelo de simulação da rede de sensores obtido a partir da aplicação da Secção 2. Na figura é possível identificar a azul os dois sensores de temperatura denominados por *Node1* e *Node2*. Estes sensores executam o código designado na Secção 2.1 pelo ficheiro node. callas , que faz com que enviem periodicamente o valor da temperatura de cada contentor para a estação-base, representada na figura a verde. Por seu lado, a estação-base, que executa o código do ficheiro sink . callas , notifica uma entidade central sempre que a temperatura em alguns dos contentores excede 10 graus centígrados. O modelo apresentado é gerado automaticamente em função da informação definida no ficheiro network. calnet . É possível observar o raio de alcance da comunicação dos sensores, bem como a sua posição relativa.

A comunicação entre os sensores e a estação-base ocorre através do canal CalasPowerLossChannel, que simula a comunicação sem fios entre os sensores no contentor. Este canal simula um meio de comunicação com perda de sinal e evita a não transmissão de mensagens repetidas. As portas in/out de todos os nós representados recebem/enviam mensagens utilizando o canal CalasPowerLossChannel. A comunicação da estação-base com um sistema central fora do camião é simu-

lada pelo canal GSMChannel. Este canal é o responsável pela ligação da simulação de sensores com a execução de fluxos de trabalho. A estação-base é a única que pode enviar mensagens no canal GSMChannel.

O fluxo de trabalho descrito pela Figura 4 calcula a melhor trajectória para as entregas, tendo em conta a temperatura do contentor e a localização do camião. A integração faz-se encapsulando o modelo da rede num actor do Kepler (TruckNetwork), que funciona como uma fonte de dados. A execução do fluxo de trabalho é, neste caso, despoletada por uma mensagem recebida do TruckNetwork. Num fluxo de trabalho diferente, a execução poderia decorrer iniciar-se por um outro evento, sendo alterada ao dar-se o evento proveniente da rede de sensores. No canto superior direito da Figura 3 encontra-se o actor WirelessToWired, que recebe as comunicações enviadas através do canal GSMChannel e as disponibiliza sob a forma de uma mensagem enviada para a porta de saída outPort do actor TruckNetwork representado na Figura 4. A mensagem da porta outPort do TruckNetwork é enviada para o actor MessageDisassembler, que dela extrai os valores de containerID e truckPosition, dados de entrada do fluxo de trabalho contido em CalculateBestPath, que calcula o melhor caminho a percorrer. Observe-se que o actor WirelessToWired (que liga a rede de sensores sem fios ao fluxo de trabalho) não está dependente da rede de sensores, nem do fluxo de trabalho; ele é somente um meio de transporte da mensagem.

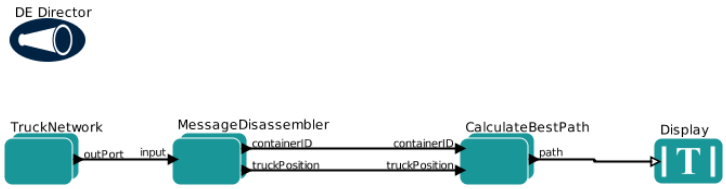


Figura 4. Fluxo de trabalho no Kepler. O actor TruckNetwork encapsula o modelo da rede da Figura 3. O actor CalculateBestPath encapsula o fluxo de trabalho de cálculo da trajectória.

As dificuldades da integração da simulação da rede de sensores no Kepler prendem-se com a (eventual) heterogeneidade dos domínios de computação. A simulação de aplicações Callas faz-se no domínio *Wireless*, uma extensão do *Discrete Event*, nem sempre adequado para a execução de fluxos de trabalho. Todavia, não deverão surgir dificuldades nos processos de negócio que estamos interessados em simular (empresariais), uma vez que são habitualmente orientados a eventos.

4 Conclusão e Trabalho Futuro

Neste artigo apresentamos uma forma de integrar a simulação de redes de sensores na execução de fluxos de trabalho, o que permite testar aplicações de mais alto nível baseadas em informação das *coisas*. A nossa proposta pretende integrar a simulação de redes de sensores da plataforma VisualSense [5] no sistema de gestão de fluxos de trabalho Kepler [4], tirando partido da interoperabilidade de componentes (actores) entre os dois sistemas, fruto de serem ambos extensões da plataforma Ptolemy II [9]. Para a criação de modelos de simulação de redes de sensores propriamente ditas, fazemos uso de um gerador (de modelos de simulação) que apresentámos anteriormente [19]. Pensamos que a nossa abordagem será uma mais valia na área de testes de aplicações (não só as baseadas de fluxos de trabalho) que dependam de valores recolhidos do ambiente porque permite validar fluxos de trabalho novos ou em que pretendemos experimentar novas variantes. Em contraste, o teste de fluxos de trabalho baseados em informação sobre execuções anteriores de fluxos de trabalho não se afigura tão flexível e completa como a que propomos.

Como trabalho futuro, identificamos desde logo a validação deste modelo de execução de fluxos de trabalho, bem como a obtenção de resultados que nos permitam avaliar a solução que propomos. Um ponto interessante que nos merece atenção no futuro é a interoperabilidade de fluxos de trabalho, isto é, a possibilidade de executar fluxos de trabalho, de uma forma distribuída, em dois ou mais sistemas de gestão de fluxos de trabalho distintos. A variedade de sistemas de gestão de fluxos de trabalho, motores e linguagens de descrição, dificulta a interoperabilidade dos fluxos de trabalho. Por exemplo, o Taverna [10] interpreta a linguagem Simple Conceptual Unified Flow Language [10] (SCUFL), o Kepler interpreta Modeling Markup Language [6] (MoML), o sistema YAWL [18] usa a linguagem Yet Another Workflow Language [17] (YAWL) e o Triana [16] interpreta, além do seu próprio formato, Business Process Execution Language [11] (BPEL). A acrescer à dificuldade decorrente da variedade de linguagens de descrição e de plataformas de execução, frequentemente as linguagens têm expressividades diferentes, pelo que a tradução entre elas nem sempre é possível, o que compromete a interoperabilidade por via da tradução de linguagens.

A interoperabilidade de fluxos de trabalho poderia ser conseguida através da padronização da linguagem de descrição/execução. Tem havido tentativas nesse sentido, por exemplo, o Workflow Management Coalition criou o XPDL [11] e a Microsoft e a IBM criaram o BPEL, ambos com o intuito de se tornarem padrão. Um outro caminho para a interoperabilidade de fluxos de trabalho é o da integração de motores de fluxos de trabalho de tal forma que seja possível correr cada fluxo no seu ambiente de execução, mas podendo interagir com outros, a correr noutros ambientes. Uma abordagem nesse sentido é apresentada por Kukla et al. [12]. Os autores vêem os sistemas de gestão de fluxos de trabalho como aplicações *legacy* embebidas num ambiente de Grid Computing, no caso, no GEMLCA [7] (Grid Execution Management for Legacy Code Applications).

Embora não tivéssemos abordado a disponibilização da informação de sensores via *web*, tal não se afigura complicado e vislumbramos que poderá ser feito facilmente com recurso a serviços *web*: teria de se criar uma interface que expu-

sesse o simulador VisualSense como serviço *web* de forma a poder ser acedido remotamente.

Agradecimentos

Os autores são parcialmente suportados pelo projecto CALLAS da Fundação para a Ciência e Tecnologia (PTDC/EIA/71462/2006).

Referências

1. G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.
2. I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.
3. I. F. Akyildiz, M. C. Vurany, B. Ozgur, and A. Weilian Su. Wireless Sensor Networks: A Survey Revisited . *Computer Networks*, 2005.
4. I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *SSDBM'04: Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, pages 423–424. IEEE Computer Society, 2004.
5. P. Baldwin, S. Kohli, E. A. Lee, X. Liu, and Y. Zhao. Modelling of sensor nets in Ptolemy II. In *Proceedings of IPSN'04*, pages 359–368. ACM Press, 2004.
6. Christopher Brooks, Edward A. Lee, Xiaojun Liu, Stephen Neuendorffer, Yang Zhao, and Haiyang Zheng. Heterogeneous concurrent modeling and design in java (volume 1: Introduction to ptolemy ii). Technical Report UCB/EECS-2008-28, Electrical Engineering and Computer Sciences University of California at Berkeley, 2008.
7. T. Delaitre, A. Goyeneche, P. Kacsuk, T. Kiss, G. Terstyanszky, and S. Winter. Gemlca: Grid execution management for legacy code architecture design. In *EUROMICRO'04: Proceedings of the 30th EUROMICRO Conference*, pages 477–483. IEEE Computer Society, 2004.
8. E. Egea-Lopez, J. Vales-Alonso, A. Martinez-Sala, P. Pavon-Mariño, and J. Garcia-Haro. Simulation Scalability Issues in Wireless Sensor Networks. *IEEE Communications Magazine*, 44(7):64–73, 2006.
9. J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the ptolemy approach. *Proceedings of IEEE*, 91(2):127–144, Jan 2003.
10. D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web Server issue):729–732, 2006.
11. R. Ko, S. Lee, and E. Lee. Business process management (bpm) standards: A survey. *Business Process Management journal*, 15(5), 2009.
12. T. Kukla, T. Kiss, G. Terstyanszky, and P. Kacsuk. A general and scalable solution for heterogeneous workflow invocation and nesting. In *WORKS 2008: Proceedings of the Workflows in Support of Large-Scale Science, Third Workshop*, pages 1–8. Springer-Verlag, 2008.
13. L. Lopes, F. Martins, and J. Barros. *Middleware for Network Eccentric and Mobile Applications*, chapter 2, pages 25–41. Springer-Verlag, 2009.

14. F. Martins, L. Lopes, and J. Barros. Towards the safe programming of wireless sensor networks. In *Proceedings of ETAPS'09*, 2009.
15. A. Rozinat, M. T. Wynn, W. M. P. van der Aalst, A. H. M. ter Hofstede, and C. J. Fidge. Workflow simulation for operational decision support. *Data Knowl. Eng.*, 68(9):834–850, 2009.
16. I. J. Taylor and B. F. Schutz. Triana - A Quicklook Data Analysis System for Gravitational Wave Detectors. In *Second Workshop on Gravitational Wave Data Analysis*, pages 229–237. Editions Frontières, 1998.
17. W. van der Aalst. Yawl: yet another workflow language. *Information Systems*, 30(4):245–275, June 2005.
18. W. M. P. van der Aalst, L. Aldred, M. Dumas, and A. H. M. Ter Hofstede. Design and implementation of the yawl system. In *CAiSE'2004*. Springer-Verlag, 2004.
19. D. Vieira and F. Martins. Automatic generation of WSN simulations: From Callas applications to VisualSense models. In *Proceedings of SENSORCOMM'2010*, 2010 (to appear).
20. E. Yoneki and J. Bacon. A survey of wireless sensor network technologies: Research trends and middleware's role. Technical Report UCAM-CL-TR646, University of Cambridge, 2005.

IoT-aware business processes for logistics: limitations of current approaches

Pedro Ferreira[1], Ricardo Martinho[1], Dulce Domingos[2]

[1] School of Technology and Management, Polytechnic Institute of Leiria, Portugal
[2] Faculty of Science, University of Lisboa, Portugal
{pedro.ferreira, ricardo.martinho}@ipleiria.pt ; dulce@di.fc.ul.pt

Abstract. The Internet of Things (IoT) aims at bridging the gap between real-world business processes and information systems. Supply chain management is one of the major application areas that can benefit from the IoT. When attached to physical items, the IoT technologies such as RFID and sensor networks transform objects of the supply chain into *smart items*. These items have the ability to capture context data and provide information systems with a representation of 'things'. This allows information systems to monitor the supply chain processes through process aware information systems. Smart items can also execute parts of the business processes. In distributed environments, they can exchange data among them and make decisions based on business logic. However, this logic only acts according to pre-planned behaviour. Unpredicted exceptions based on real life events require dynamic process adaption in process definitions and corresponding instances. In this paper we review the main technologies of the IoT associated with automated support of business processes in logistics. We also identify the main limitations in the Business Process Execution Language (BPEL), regarding the support of design and runtime changes in these processes with smart items.

Keywords: The Internet of Things, smart items, logistics, business process, flexibility, BPEL.

1 Introduction

In the last decade, the term Internet of Things (IoT) has been raising interest on the enterprise world, mostly due to a growing web-based service economy [1]. The IoT provides a key role for the future Internet by bridging the gap between the physical world and its representation in information systems.

From an enterprise point of view, manufacturing, supply chain integrity, energy, health and automotive are some of the major application areas of the IoT. Despite the benefits, major technical issues such as internet scalability, identification and addressing, heterogeneity and service paradigms are prominent areas of research in recent years [2].

The supply chain is a network of organisations and business processes for procuring raw materials, transforming them into products and distributing these to

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 611–622

customers. There are five major supply chain processes: plan, source, make, deliver and return [3]. Logistics plays an important role in these processes, dealing with the control and planning of all the factors that will have an impact on transporting [3].

Technologies, such as RFID and sensors, provide context data to support decision making at high management level. The introduction of sensors with the ability to execute business logic at the item level, *i.e., smart items* [4], allows local decision-making and therefore reduces centralised processing and the amount of exchanged data. The decomposition of business processes through distributed environments creates a paradigm shift and challenges for business process modelling.

The Business Process Execution Language for Web-Services (WS-BPEL) has emerged as the standard reference to model the behaviour of Executable and Abstract business processes on Web Services [5]. It defines an interoperable integration model, extending the Web Services interaction model and enabling it to support business transactions.

So far it is possible to use information provided by the IoT to support static business processes, *i.e.*, processes defined at design time that do not foresee deviations. However, the use of smart items often requires dynamic business processes that are able to accommodate adaptations, according to changes verified in the execution context or behaviour of smart items.

In this paper, we present the limitations of BPEL to define business processes that support this dynamic behaviour. We also address how this ability can dictate the way business process logic is distributed between smart items and standard process support. We focus on logistics and supply chain-related business processes, which make use of smart items.

The remainder of this paper is organized as follows: section 2 describes the influence of smart items at supply chain; section 3 analyses the impact of business logic at smart items; section 4 discusses the limitations of modelling business processes with BPEL and section 5 concludes this paper.

2 Smart items in logistics

The main purpose of the IoT is to fill in the gap that usually exists between real-world business processes and their representation within information systems. Therefore, technologies such as RFID and wireless sensor networks capture accurate context data. These data can then be used in real-time representations of business processes and involved objects within information systems. For these purposes, the technologies and related devices are commonly referred to as *smart items*.

2.1 Smart items types

Three main technologies are commonly used by smart items in business processes related with logistics. They are *barcodes*, RFID and *sensor networks*. Barcodes are a standard technology for electronic identification of products. A barcode is attached on the product and optically detected by a barcode reader. The reader acknowledges the printed identification and provides acquired data to the information system, which

updates the product's state. This solution provides limited information due to line-of-sight requirement. For instance, it is impossible to detect a single item within a closed container of products. The acquisition of product's data during transportation requires a more complex infrastructure. Therefore barcodes are only useful in load and unload processes within the logistics of the supply chain.

Radio Frequency Identification (RFID) devices [6] can be identified through radio-based frequency handling technologies. Unlike barcodes, they do not require line-of-sight to be identified. Location tracking is available including products in transportation, depending on RFID readers deployed. They also have the ability to acquire products sensor data (such as temperature) and provide it to the information system. These sensing capabilities are usually very limited [7]. Accordingly to their behaviour, barcodes and RFID can be referred to as *passive* devices [8].

Wireless sensor networks are the most promising approach for logistics processes. The sensor nodes are electronic devices with embedded sensing and computing systems that collaborate within a network. In addition, they are extremely small and can be specifically designed to meet the requirements of the transported products. Unlike RFID, sensor networks can execute parts of the processes from an information system directly on the items. Products become embedded logistics information systems [7]. Sensor networks cover identification, tracing, location tracking, monitoring and real-time responsiveness. For instance, CoBIs [9] presents a sensor network that covers all these aspects.

2.2 Logistic functions, information systems and smart items

Logistic functions can be associated with a set of features commonly supported by IS and smart items' technologies, as illustrated in Table 1. The basic logistics functions are to transport "*the right goods and the right quantity and right quality at the right time to the right place for the right price*" [7]. To address each of these functions, information systems must have specific features, such as *identification*, *tracing*, *location tracking*, *monitoring*, *real-time responsiveness* and *optimisation*. Product identification informs the system about the *right goods*. Tracing allows the system to detect when items are lost. Therefore, it guarantees the *right quantity*. The *right place* is monitored by the information system through location tracking. It keeps track of the transport itself. Monitoring the product's state ensures the product's *right quality*. With all these data within the information system, the overall logistics process can be observed with detail. Therefore, responsiveness to unforeseen events and other actions can be achieved at the *right time*. In addition, these data provide the basis for optimisation affecting the product's *right price*. Smart items play an important role in supporting all of these features. Moreover, the kind of support provided to these features can be directly correlated with the types of smart items referred in previous section.

Table 1. Logistic functions and information systems features to implement them. It also displays the smart items capabilities towards logistic functions.

Functions	Features	Barcode	RFID	Sensor Networks
right goods	Identification	Full	Full	Full
right amount	Tracing	Partial	Full	Full
right place	Location Tracking		Full	Full
right quality	Monitoring		Partial	Full
right time	Real-time responsiveness			Full
right price	Optimisation			Full

2.3 The role of smart items in supply chain integrity

The information generated by smart items allows the monitoring and control of products, along the entire logistics process. For instance, tracking their location can be used to detect if the associated products have been detoured from a pre-planned route. Tracking their state can be used to realise if the products' condition has changed and whether they are still useful or not. This denotes three types of integrity that can be compromised: (1) product integrity, (2) components integrity, (3) route integrity.

For instance, a product's physical integrity can be monitored through sensors that keep status during its life-cycle within the logistics process. As an example, we can consider that a product starts the process tagged with the status *closed*. If this status changes during the process (to *opened* or *tampered*), the product's integrity might have been compromised. For perishable products, sensors with the ability to measure temperature can be used to monitor the product's condition. For example, a truck loaded with fruit starts the process tagged with the status *good*. If the temperature rises above a pre-planned threshold value, the fruit's quality may be affected [8].

Regarding transportation routes integrity, these can be monitored through technologies that provide products location. Transporting these products requires pre-planned routes. However, there may be detours from the original route due to environment changes. For example, when dealing with hazardous products, some restrictions can apply to available routes and they can even be unauthorized. Unpredicted events such as traffic, weather conditions or road blocks might compromise the products route integrity during transportation.

Integrity of components requires the most complex monitoring. It consists on controlling every component of the product during its production and transportation. It ensures that the product keeps its intended use and does not break established rules regarding legal issues or environmental compliance along the logistics process.

In addition, breaking one of these integrity types can result in also affecting the other remaining ones. For instance, the fruit truck may have to detour due to a product integrity breach. Conversely, the product's integrity can be affected due to a forced detour.

3 Business process logic within smart items

Smart items provide new opportunities and challenges in the system design and electronics integration. Based on their potential and collaboration with external services, they are able to do more than providing real-time data. They also process data and make decisions based on it, including exchanging data among smart items that do not depend on a centralised model. In this section, we refer to the software architectures commonly used to address these issues. We also present relevant factors that affect the amount of business logic that is distributed between a central system and cooperative smart items.

3.1 Architectural evolution

The Internet of Things is a concept constantly evolving mostly due to its young existence. It first appeared with the use of RFID and evolved through related technologies such as sensor networks and smart embedded devices. The introduction of smart items at the supply chain logistics processes requires constant optimisation and innovation, in order to enhance enterprises' competitiveness and quality of service.

Architectures used to accommodate interactions between smart items and information systems have also been evolving at a similar pace.

For instance, client-server architectures still play a major role regarding these interactions. Nevertheless, Service Oriented Architectures (SOA) is becoming the preferred approach regarding interactions of information systems with more powerful smart items. Moreover, this is indicated as the dominant architectural approach for these kinds of devices in the future [12].

The integration of smart items into business processes through SOA allows information systems to interact with physical objects and create *the Internet of Services* (IoS). This integration is possible by running instances of web services on these devices. Such architectural change provides an outnumbered set of opportunities and challenges in achieving efficient collaboration between the services and centralised information systems. In order to overcome these challenge, middleware approaches have been a reliable solution to integrate back end applications and services offered by the devices, service-mediators and gateways [12].

The SIRENA (Service Infrastructure for Real-time Embedded Networked Applications) [10] project was developed to leverage SOA architectures to seamlessly connect embedded devices within different domains. This project presents proof-of-concepts that illustrate the feasibility and benefits of embedding web services at devices. However, these pioneer efforts lacked attention to issues such as device supervision, device life cycle management or maintaining the status of discovered devices. SIRENA was also used as a foundation for SODA [11] and SOCRADES [12] projects. The purpose of SODA was to create a comprehensive, scalable, easy to deploy, service-oriented ecosystem built on top of foundations laid by SIRENA. This project led to significant reduction of time to market for innovative services. The purpose of SOCRADES was to develop a design, execution and management platform, exploiting the SOA paradigm at device and application level. The

SOCRADES middleware is an architecture that provides web service enabled devices for business integration with information systems such as ERPs for the manufacturing area.

3.2 Delegating business logic to smart items

As described in section 2, there are different types of smart items according to their behaviour and characteristics. Therefore, some authors fit smart items in two different groups: *passive* and *active*. Passive technologies such as RFID and barcodes can identify products at transhipment points. Semi-passive RFID data loggers allow temperature recording at affordable costs. Active technologies such as wireless sensor networks can communicate among all participants in a supply chain's logistic process (freight, containers, warehouses and vehicles).

Böse and Windt presented a catalogue of thirteen criteria to characterise logistic systems regarding autonomy [13]. The location of the decision making is considered the most important criterion concerning autonomous control. Despite of having an essential role on monitoring the process, smart items have been mostly used as information providers instead of participants in decision making or business process planning. The idea of delegating some business logic to smart items shifts the decision making from centralised, server-based solutions to a network of distributed processing devices. This creates an autonomous cooperation within the logistics business processes. Each smart item has its own piece of software, which can autonomously search for a partial solution when dealing with process-related issues. In transportation scenarios, this software collects information, makes decisions and negotiates with other entities to fulfil their goal. For instance, a truck loaded with several pallets of fruit can have each one equipped with a smart item. These can monitor a physical dimension such as temperature, which in turn will dictate the truck route in order to deliver all products at the minimal costs [8].

In controlled transportation scenarios, *i.e.*, not subjected to unforeseen circumstances, everything is determined before the process begins. Therefore, there is no need for delegating new behaviour to the smart item level. However, changes in traffic, new incoming orders, lack of communications with the central system or any other kind of unforeseen events might require a detour to a pre-planned route. To support these unexpected scenarios, it is necessary to use smart items with embedded intelligence enough to provide for dynamic planning.

This approach may require a shift of the business logic and associated control from the central system level to the smart item level. From this point of view, decisions are made in real-time by smart items on the field using their interaction abilities and intelligence, without human direct intervention. Therefore, in order to keep the system running, the implementation of software to be embedded into the smart item must provide robustness, flexibility, privacy, low communication costs and low computation time.

Supply chain management systems equipped with these smart items must have flexibility to react immediately to sudden changes. For instance, if a road block happens in a transportation scenario, the best alternative route must be searched immediately. This route must be in accordance with the logistic functions and keep

supply chain integrity (referred above in section 2). The need of fast responsiveness requires low computation time.

However, a thorough search for optimal route in a complex scenario could take too much time depending on the smart items processing capabilities. If a communication failure occurs in the network, system should be robust enough to continue its work. Internal planning strategies and other sensitive data must be kept confidential. For instance, if a route change is necessary, the delivery time for several customers will most likely change. Despite of being aware of this change, each customer must not have access to other customer's changes [2].

In a central based approach, objects in the logistics process are simply information providers. Therefore they only execute atomic activities defined in a business process running on a central system [12]. Smart items with embedded intelligence handle incoming data, observe and evaluate surrounding conditions and make decisions based on acquired information. However, these depend on the objects decision freedom within the process and, consequently, their ability for process dynamic changes [8].

4 Dynamic changes to business processes with BPEL

Changing business processes dynamically involves altering the process's control flow, data or resource perspectives at runtime. Examples include adding, skipping, updating or deleting an activity, changing the data objects associated with an activity, or even altering its role-assignment. However, these changes must assure the correctness (syntax) of process definitions and process instances, and consistency among concurrently executed process instances [14]. Therefore, *flexibility* has been an issue concerning the business process management and workflow research areas.

4.1 Process flexibility types

After several case studies and years of research, consensus was obtained concerning the flexibility required to deal with exceptions. Eder and Liebhart [17] grouped exceptions into two groups: predicted and unpredicted. Predicted exceptions represent the unusual but foreseen behaviour of a process. These exceptions can be modelled in the process definition as alternative paths to normal behaviour. The unpredicted exceptions represent the unforeseen behaviour of a real world business process regarding the process definition. To address these unpredicted exceptions, systems need to update the process definition and the corresponding process instances.

In a sequence of also recent contributions, Schonenberg et al. present a taxonomy of process flexibility [15]. Four distinct types to process flexibility are identified, each having its own application area. We enumerate them below, referring a simple transportation process scenario for each one of them:

- *Design*: for handling anticipated changes in the operating environment, where supporting strategies can be defined at design-time;

- *Deviation*: for handling occasional unforeseen behaviour, where differences with the expected behaviour are minimal;
- *Underspecification*: for handling anticipated changes in the operating environment, where strategies cannot be defined at design-time, because the final strategy is not known in advance or is not generally applicable;
- *Change*: either for handling occasional unforeseen behaviour, where differences require process adaptations, or for handling permanent unforeseen behaviour.

Each of the flexibility types operates in different ways. Figure 1 provides an illustration of the distinction between each of the flexibility types in isolation, in terms of the time that specific flexibility options need to be configured - at design time, as part of the process definition or at runtime via facilities in the process execution environment. It also shows the anticipated completeness of the process definition for each flexibility type.

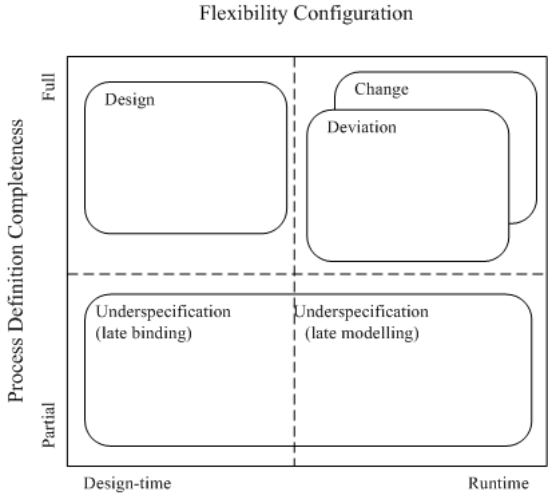


Fig. 1. Taxonomy of process flexibility according to Schonenberg et al. (adapted from [15]).

4.2 BPEL limitations to process flexibility

So far we have described the types of smart items and how they can benefit business processes in logistics. We have also observed the delegation of business logic to smart items due to architectural evolution. This evolution also allows the decomposition of business processes through distributed networks instead of central based solutions. However, none of these approaches supports flexibility in process that also includes smart items. This means that these business processes do not foresee either predicted or unpredicted changes that may force updates in the business logic running on both central system and smart items.

As referred above, WS-BPEL has emerged as a standard reference language for modelling and executing business processes. A WS-BPEL process definition includes partner links that define the relationships with other business partners, declarations of process data, handlers for various purposes and the activities. Basic activities only perform their intended purpose, such as receiving a message from a partner, or manipulating data. Structured activities can contain other activities and define the control flow business logic (see [5] for further details).

We foresee its application also in business processes for logistics that use smart items. However, and as we already referred, this kind of processes can be subjected to many predicted and unpredicted changes. For this matter, WS-BPEL has some limitations, and we will classify them according with the flexibility types illustrated in Figure 1.

Regarding flexibility in *design time*, we can identify the following limitations when dealing with WS-BPEL:

- The definition of alternative flows is possible but limited concerning the number of paths – WS-BPEL allows the handling of predicted exceptions with exception handlers, as well as alternative flows through the use of if/else control structures (flexibility by *design*). However, WS-BPEL fails in allowing for a compact process definition for a larger number of exceptions and alternative paths, which cannot be foreseen or practically defined. In the IoT context, business processes definitions that rely on smart items' collected data may imply a large number of exception handlers or alternative paths;
- All process perspectives (control flow, data types and handlers) must be defined in a static way and *a priori* – WS-BPEL does not allow for flexibility by *underspecification*, meaning that process definitions cannot be partially defined or incomplete, or even dynamically specified (e.g., it is not possible to provide a partner link's name later on when the process as already began to execute);
- The definition of business logic that is to be run in smart items is not possible with WS-BPEL. It would be valuable to access and specify all sub-process definitions that compose a business process model together. This may include definitions of the business process logic to be executed either centrally or on the smart items. WS-BPEL provides extension mechanisms that can be used to define additional language constructs, in order to model the business logic to be loaded into the smart items;
- WS-BPEL does not foresee the distribution of business logic between a central system and smart items, according to smart items properties. We must keep in mind that smart items are electronic devices. Therefore, they have physical properties such as power (batteries) and computation speed that limit their autonomy. When delegating business logic into smart items, processing is required. As more of the business process is delegated, the more processing will be necessary. Therefore, power consumption will slightly increase. On the other hand, the less of the business process is delegated to smart items, the more communication will be required, highly increasing power consumption. In addition, smart items can have different

capabilities. Therefore they can support distinct amounts and types of business process logic;

- WS-BPEL does not allow controlling which, how and by whom parts of a process definition can be changed. This *controlled flexibility* [20] can be useful for our context, specifically for clarifying which parts of a process can be changed, when the process is distributed between the central system and the smart items;

As for flexibility in runtime, the major limitations that we can identify in WS-BPEL are:

- The lack of support for changes of business process instances, due to unpredicted, ad-hoc circumstances. Logistics with smart items are subjected to a plethora of these circumstances, which generate events that must be immediately reflected in changes in the governing business processes.
- The lack of support in migrating instances from old process definitions to new ones, when a redefinition of the business process occurs. Some works have already addressed these challenges in WS-BPEL, including correctness and compliance issues (e.g., see [14], [18]), but out of the context of the IoT. However, it is possible to redefine smart items behaviour in runtime, for example using the Callas language presented in [19].

Moreover, combining these types of flexibility adds specific challenges, also not addressed by WS-BPEL. These include the use of runtime ad-hoc changes and design changes together. Reichert *et al.* allow changes to be propagated to the process instances, which were already subjected to ad-hoc changes [16]. Also, the use of design changes together with underspecification flexibility in runtime (late binding) raise additional challenges regarding correctness and compliance between process definitions and underspecified process running instances.

5 Conclusions

The IoT is a concept raising interest in logistics business processes, mostly due to use of technology commonly referred to as smart items. These items provide accurate context data to information systems, which they use in real-time representations of business processes. Smart items like wireless sensor networks with embedded computing systems can do more than just providing data. They can execute parts of business processes and cover the basic logistics functions.

Central based solutions still play an important role in logistics processes; however distributed solutions are becoming the preferred approach. The introduction of sensors with the ability to execute business logic at the item level allows local decision-making and therefore reduces centralised processing and the amount of exchanged data. However, none of these approaches supports predicted or unpredicted changes that can occur in real world business processes. These changes require business processes to be redefined or process instances to be changed handled dynamically, including changes in the process control flow, data and resources at runtime, such as reprogramming the smart items.

Summing up, we stressed WS-BPEL's limitations on process flexibility and classified them according to the taxonomy types defined in Section 4.1. These limitations shed some light on future topics that we intent to explore on WS-BPEL, namely allowing for the definition and distribution of business process logic between central systems and smart items. Also, we are already addressing some of these challenges through an extension language for WS-BPEL regarding the definition of smart items business logic. For this we are taking advantage on the native extension mechanisms on WS-BPEL, specifically through the `<extensionActivity>` element.

Acknowledgments

This work was supported by FCT through project PATI (PTDC/EIA-EIA/103751/2008) and through LASIGE Multiannual Funding Programme.

References

1. ISTAG Working Group Report on "Web-based Service Industry", February 2008, ftp://ftp.cordis.europa.eu/pub/ist/docs/web-based-service-industry-istag_en.pdf
2. Haller, S., Karnouskos, S., Schroth, C.: The Internet of Things in an Enterprise Context. In Future Internet — FIS 2008: First Future Internet Symposium, FIS 2008 Vienna, Austria, September 29-30, 2008 Revised Selected Papers, pages 14–28, Berlin, Heidelberg, 2009. Springer-Verlag.
3. Laudon, K.C., Laudon, J. P.: Management Information Systems. New Jersey: Pearson Prentice Hall, 385--389 (2006).
4. Uckelmann D.: A Definition Approach to Smart Logistics. S. Balandin et al. (Eds.): Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN 2008), number 5174 in LNCS, pages 273-284, Springer-Verlag, 2008
5. OASIS. Web Services Business Process Execution Language (WS-BPEL), Version 2.0. Technical report, Organization for the Advancement of Structured Information Strandards, 2007.
6. Finkenzeller, K.: RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification (2003)
7. Decker, C., Berchtold, M., Chaves, L., Beigl, M., Roehr, D., Reidel, T., Beuster, M., Herzog, T., Herzig, D.: Cost-Benefit Model for Smart Items in the Supply Chain. In C. Floerkemeier and et Al., editors, Proceedings of The Internet of Things. First International Conference (IOT 2008), number 4952 in LNCS, pages 155-172. Springer-Verlag, 2008.
8. Jedermann, R., Lang, W.: The benefits of embedded intelligence - tasks and applications for ubiquitous computing in logistics. In C. Floerkemeier and et Al., editors, Proceedings of The Internet of Things. First International Conference (IOT 2008), number 4952 in LNCS, pages 105–122. Springer-Verlag, 2008.
9. Decker, C.,Reidel, T., Beigl, M., sa de Souza, L.M., Spiess, P., Mueller, J., Haller, S.: Collaborative Business Items. In 3rd International Conference on Intelligent Environments (2007)
10. Bohn, H., Bobek, A., Golatowski, F.: SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains, icniconsmcl, pp.43, International Conference on Networking, International Conference on

Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06), 2006
11. Deugd, S., Carroll, R., Kelly, K., Millett, B., Ricker, J.: SODA: Service Oriented Device Architecture, IEEE Pervasive Computing, vol. 5, no. 3, pp. 94-96, c3, July-Sept. 2006
12. Souza, L., Spiess, P., Guinard, D., Köhler, M. Karnouskos, S., Savio, D.: SOCRADES: A Web Service based Shop Floor Integration Infrastructure. In C. Floerkemeier and et Al., editors, Proceedings of The Internet of Things. First International Conference (IOT 2008), number 4952 in LNCS, pages 50–67. Springer-Verlag, 2008.
13. Böse, F., Windt, K.: Catalogue of Criteria for Autonomous Control. In: Hülsmann, M., Windt, K. (eds.) Understanding Autonomous Cooperation and Control in Logistics – The Impact on Management, Information and Communication and Material Flow, pp. 57–72. Springer, Berlin (2007)
14. Reichert, M., Rinderle, S.: On design principles for realizing adaptive service flows with bpel. In Proceedings of International Conference on Conceptual Modeling (EMISA 2006), pages 133–146. Lectures Notes in Informatics, 2006
15. Schonenberg, H.; Mans, R.; Russell, N.; Mulyar, N. & van der Aalst, W. M. P. Towards a Taxonomy of Process Flexibility Proceedings of the Forum held at the 20th Conference on Advanced Information Systems Engineering (CAiSE'08), 2008
16. Reichert, M., Hensinger, C., & Dadam, P. (1998). Supporting Adaptive Workflows in Advanced Application Environments. In Proceedings of the EDBT Workshop on Workflow Management Systems. Valencia, Spain.
17. Eder, J., Liebhart, W.: The workflow activity model WAMO. In: Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIS'95), Vienna, Austria, May 1995, pp. 87–98
18. Fang, R., Zou, Z. L., Stratan, C., Fong, L., Marston, D., Lam, L., Frank, D.: Dynamic Support for BPEL Process Instance Adaptation. In Proceeding of the 2088 IEE International Conference on Services Computing, 2008, pp. 327–334
19. Martins, F., Lopes, L., Barros, J.: Towards Safe Programming of Wireless Sensor Networks. In Proceedings of Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES), 2010
20. Martinho, R.; Varajão, J. & Domingos, D.: Modelling and Learning Controlled Flexibility in Software Processes. *International Journal on Knowledge and Learning*, 2009, 5, 423-442

Segurança de Sistemas de Computadores e Comunicações

Melhorando a Fiabilidade e Segurança do Armazenamento em *Clouds*

Bruno Quaresma, Alysson Bessani, and Paulo Sousa

Laboratório de Sistemas Informáticos de Grande Escala,
Faculdade de Ciências da Universidade de Lisboa,
Lisboa, Portugal

Abstract. With the increasing popularity of *cloud* storage services, companies that deal with critical data start thinking of using these services to store medical records databases, historical data of critical infrastructures, financial data, among others. However, many people believe that information stored that way is vulnerable, despite the guarantees given by providers, which makes reliability and security the major concerns about *cloud* storaging. In this work we present DEPSKY, a system that improves the availability, integrity and confidentiality of information stored in the *cloud*.

Resumo. Com a crescente popularidade das *clouds* de armazenamento, empresas que lidam com dados críticos começam a pensar em usar estes serviços para armazenar bases de dados de registos médicos, históricos de infra-estruturas críticas, dados financeiros, entre outros. No entanto, muitas pessoas acreditam que informação armazenada num sistema deste tipo é vulnerável, apesar de todas as garantias dadas pelos fornecedores, o que faz da fiabilidade e da segurança as maiores preocupações sobre o armazenamento em *clouds*. Neste trabalho apresentamos o DEPSKY, um sistema que melhora a disponibilidade, integridade e confidencialidade de informação armazenada na *cloud*.

1 Introdução

Actualmente, muitas organizações começam a optar de forma progressiva pelo uso de *clouds* de armazenamento. Exemplos recentes são serviços como o Twitter e o Facebook que até há bem pouco tempo tinham os seus próprios *data centers* de armazenamento e hoje terceirizam parte deste serviço para a Amazon e o seu Simple Storage Service (Amazon S3) [1]. Esta tendência pode ser definida como o armazenamento de informação num sistema de armazenamento remoto mantido por terceiros. A Internet fornece a ligação entre o computador e esse sistema.

O armazenamento em *clouds* tem algumas vantagens sobre o armazenamento tradicional. Por exemplo, se se armazenar informação numa *cloud*, esta estará acessível a partir de qualquer local com acesso à Internet e evita a necessidade da manutenção de uma infra-estrutura de armazenamento (e.g., uma rede de discos) na organização. Além disso, o modelo de cobrança das *clouds* de armazenamento incorpora o conceito de elasticidade de recursos: paga-se apenas pelo uso e o serviço pode crescer arbitrariamente para tolerar altos picos de carga esporádicos.

À medida que as *clouds* de armazenamento se tornam mais e mais populares, empresas que lidam com dados críticos começam a pensar em usar estes serviços para

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 625–636

armazenar bases de dados de registos médicos, históricos de infra-estruturas críticas, dados financeiros, entre outros. No entanto, um perigo muitas vezes ignorado está no facto dos sistemas de armazenamento remoto estarem fora do controlo dos donos dos dados, apesar das garantias dadas pelos fornecedores (e.g., SLAs de serviço), o que faz da fiabilidade e da segurança as maiores preocupações sobre o armazenamento em *clouds*. Neste trabalho apresentamos o DEPSKY, um sistema que garante disponibilidade, integridade e confidencialidade de informação armazenada na *cloud*. A ideia fundamental deste sistema é replicar a informação por várias *clouds* de armazenamento, utilizando algoritmos para armazenamento fiável e partilha de segredo. Além disso, apresentamos resultados experimentais que demonstram a viabilidade económica (especialmente para cargas de trabalho com bastantes mais leituras que escritas) e os benefícios em termos de desempenho e disponibilidade do sistema.

2 Clouds de Armazenamento

Existem diversos sistemas de armazenamento em *clouds*, uns possuem um foco muito específico, como armazenar apenas mensagens de e-mail ou imagens digitais, outros podem armazenar todo o tipo de dados. O seu funcionamento pode ser descrito da seguinte forma: um cliente envia ficheiros através da Internet para os servidores que guardam a informação. O acesso aos servidores pelo cliente é efectuado através de interfaces web ou serviços web (usualmente baseados no modelo REST - *REpresentational State Transfer*), que permitem o acesso e manipulação dos dados armazenados.

Nem todos os clientes estão preocupados apenas com a falta de espaço, alguns usam sistemas de armazenamento em *clouds* para *backup* de informação, o que garante que, caso haja algum problema na infra-estrutura computacional do cliente, a informação estará intacta na *cloud* de armazenamento.

Existem fornecedores de armazenamento em *clouds* que cobram uma quantia fixa por uma quota de espaço e largura de banda de entrada e saída de dados (ex., DivShare [3], DocStoc [4] e Box.net [2]), enquanto outros usam um modelo *pay-per-use* e cobram quantias variáveis consoante o espaço ocupado e a largura de banda utilizada pelo cliente (ex., Amazon S3 [1], Nirvanix [7], Windows Azure [6] e RackSpace [8]). Em geral, o preço do armazenamento *online* tem vindo baixar devido à entrada de cada vez mais empresas neste negócio.

As duas maiores preocupações acerca do armazenamento em *clouds* são a fiabilidade e a segurança. É improvável que uma organização confie os seus dados críticos a outra entidade sem a garantia que terá acesso a estes dados sempre que quiser (disponibilidade), que estes não serão corrompidos (integridade) e que mais ninguém terá acesso a eles sem a sua autorização (confidencialidade). Para garantir a segurança da informação, a maioria dos sistemas usa uma combinação de técnicas, incluindo:

- *Criptografia*: algoritmos criptográficos são usados para codificar a informação tornando-a ininteligível e quase impossível de decifrar sem a chave usada para cifrar a informação, normalmente uma chave secreta partilhada entre cliente e o serviço;
- *Autenticação*: é necessário o registo de um cliente através da criação de credenciais de acesso (ex., *username* e *password*);
- *Autorização*: o cliente define quem pode aceder à sua informação.

Mesmo com estas medidas de protecção, muitas pessoas acreditam que a informação armazenada num sistema de armazenamento remoto é vulnerável. Existe sempre a possibilidade de um *hacker* malicioso, de alguma maneira, ganhar acesso à informação do sistema, por exemplo, devido a vulnerabilidades existentes neste. Além disso, há sempre a preocupação de colocar os dados críticos (e muitas vezes confidenciais) nas mãos de terceiros, que terão acesso às informações neles contidos.

Finalmente, há também a questão da fiabilidade e disponibilidade dos serviços de armazenamento. Armazenar informação num sistema remoto acedido via Internet coloca a organização vulnerável a todos os problemas de conectividade e indisponibilidade temporária da Internet. Além disso, praticamente todos os grandes fornecedores de serviços de armazenamento já sofreram problemas de disponibilidade e/ou corromperam dados de clientes, mesmo com a redundância interna de seus sistemas (os dados são tipicamente armazenados em diferentes *data centers* do fornecedor).

3 DEPSKY

Nesta secção é apresentado o DEPSKY, um sistema para replicação de dados que melhora a fiabilidade e segurança da informação armazenada em *clouds*.

Os blocos atómicos de dados no DEPSKY designam-se por *unidades de dados* (*data units*), que podem ser actualizadas pelos seus donos e acedidas por um conjunto arbitrário de leitores. A disponibilidade destas unidades é garantida mesmo em caso de falhas devido ao uso de algoritmos de replicação para sistemas de quóruns bizantinos de disseminação [14], onde os dados armazenados em cada servidor (i.e., que neste caso são *clouds* de armazenamentos) são auto-verificáveis graças ao uso de assinaturas digitais e resumos criptográficos (i.e., se um servidor alterar o conteúdo dos dados, o leitor descobre e ignora os dados corrompidos).

O DEPSKY oferece também a possibilidade da informação mais sensível ser protegida através de um esquema de partilha de segredos [16], introduzindo garantias de confidencialidade: nenhuma *cloud*, individualmente, tem acesso à informação contida nos dados.

3.1 Modelo de Sistema

O modelo de sistema utilizado no DEPSKY segue uma série de hipóteses pragmáticas tidas em conta no desenho dos protocolos de replicação em *clouds*.

Cada *cloud* é representada por um servidor passivo (não executa nenhum código dos protocolos) que oferece operações de leitura e escrita de dados com semântica de consistência regular [12]: uma operação de leitura executada concorrentemente com uma operação de escrita retorna o valor da unidade de dados antes da escrita ou o valor que está a ser escrito.

Assumimos também que para cada unidade de dados há *apenas um escritor*, e este escritor só sofre *falhas por paragem*. Isto significa que cada bloco de dados é escrito por uma única entidade[1], o que simplifica os protocolos (que não têm de lidar com escritas concorrentes). Além disso, escritores maliciosos não são considerados pois estes

[1] Na prática pode existir mais de um escritor para uma unidade de dados desde que os acessos de escrita sejam feitos isoladamente (o que pode requerer algum controlo de concorrência).

poderiam escrever dados sem sentido do ponto de vista da aplicação de qualquer forma. Finalmente, estas duas hipóteses permitem a concretização de protocolos de leitura e escrita em sistemas onde os servidores são apenas discos passivos, como as *clouds* de armazenamento.

Os servidores (*clouds*) e leitores estão sujeitos a faltas arbitrárias ou bizantinas [13]. Da mesma forma, como são suportados múltiplos leitores para cada unidade de dados, é conveniente também assumir que estes podem ter qualquer comportamento. Devido ao uso de sistemas de quóruns bizantinos de disseminação [14], o sistema requer $n \geq 3f + 1$ servidores para tolerar até f servidores faltosos. O uso de sistemas de quóruns bizantinos também permite que o sistema tolere um número ilimitado de leitores faltosos.

Finalmente, assumimos a ausência de um sistema de distribuição de chaves entre os clientes. Os leitores apenas sabem como aceder ao sistema para ler dados e para isso possuem a chave pública do escritor para verificação e validação de dados.

3.2 Modelo de Dados

A figura 1 apresenta o modelo de dados do DEPSKY em três níveis. Num nível conceptual temos os blocos representados por *unidades de dados* (*data units*) que contêm, além do seu valor, um número de versão e informações de verificação que tornam os dados auto-verificáveis. Genericamente, uma unidade de dados do DEPSKY é representada em cada *cloud* por dois ficheiros: um contendo os metadados e o outro com o valor mais recente armazenando na unidade. Estes dois ficheiros estão sempre dentro de um *container*. O *container* de uma unidade de dados, para além de conter os metadados e o valor actual, pode conter também versões anteriores do valor desta unidade. Cada unidade de dados tem um nome único que é o seu identificador. Este é usado para obter referências para o *container* e metadados dessas unidades nos protocolos definidos.

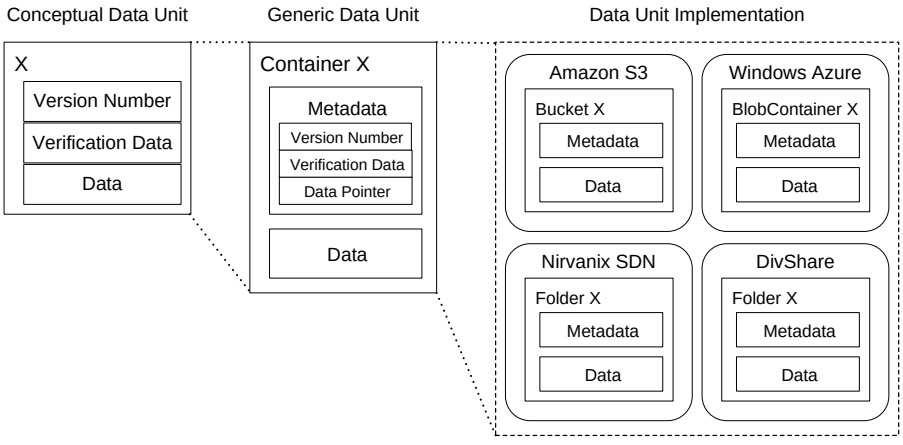


Figura 1. Decomposição do Data Unit X do DEPSKY, do conceito à concretização.

Os ficheiros de metadados são os mais importantes pois é sempre necessário um quórum destes nos protocolos definidos. Os metadados consistem na seguinte informação: um número de versão (*Version Number*), uma referência para o ficheiro com o valor desta versão (*Data Pointer*) e informação de verificação (*Verification Data*), que inclui um resumo criptográfico do valor para verificação de integridade deste e, no caso de ser uma unidade de dados com confidencialidade, dados públicos necessários para a leitura do valor. Para escrever ou ler uma unidade de dados é sempre necessário efectuar o *download* do ficheiro de metadados associado a esta em primeiro lugar.

3.3 ADS - *Available* DEPSKY

Esta secção apresenta o algoritmo ADS que promove uma melhoria da disponibilidade de dados na *cloud* através da replicação das unidades de dados por várias *clouds* de armazenamento.

Algoritmo de escrita.

1. Um cliente escritor começa por enviar um pedido de leitura dos metadados a todas as *clouds*. O escritor espera $n - f$ ficheiros de metadados correctamente assinados por ele e lidos de diferentes *clouds* para então obter o número de versão máximo dentre os contidos nestes ficheiros.
2. O número de versão lido no passo anterior é incrementado em uma unidade, dando origem ao número de versão dos dados a serem escritos nesta operação. Um ficheiro a conter os dados a serem escritos e cujo nome corresponde ao nome da unidade de dados concatenado com o número de versão é criado em todas as *clouds*. O escritor espera confirmação da escrita deste ficheiro de $n - f$ *clouds*.
3. Após conclusão da escrita da nova versão, são actualizados os metadados para a nova versão sendo enviados pedidos de escrita para este efeito. Neste passo o ficheiro de metadados é actualizado (ficheiro com metadados anterior é sobrescrito), ao contrário do passo 2 em que é escrita uma nova versão dos dados num ficheiro diferente do da versão anterior. A operação de escrita termina quando se recebe confirmação da actualização de metadados de $n - f$ *clouds*.

É importante referir que o algoritmo de escrita preserva as versões anteriores da unidade de dados. Estas versões podem ser apagadas quando o escritor achar conveniente através de um procedimento de *garbage collection* que envia pedidos de remoção suportados por todas as *clouds* estudadas.

Algoritmo de leitura.

1. Um cliente leitor começa por efectuar pedidos pelos metadados a todas as *clouds* e esperar por $n - f$ ficheiros de metadados correctamente assinados pelo escritor. O leitor obtém o número de versão máximo reportado nestes ficheiros.
2. Após obter o número de versão mais actual da unidade de dados, o cliente envia pedidos de leitura para esta versão a todas as *clouds* e espera a recepção de um valor cujo seu resumo criptográfico seja igual ao resumo criptográfico contido nos metadados, sendo então a operação terminada e este valor retornado.

Optimização de leitura. Uma optimização importante para diminuir os custos monetários do protocolo de leitura (ver secção 5.1) é enviar o pedido de leitura da versão mais actual do valor da unidade de dados apenas à *cloud* que responde mais rapidamente à requisição de metadados e reportar a versão mais actual dos dados. Desta forma, no melhor caso (sem falhas), apenas uma das *clouds* será lida. Caso esta *cloud* não responda atempadamente, outras *clouds* são acedidas até que se obtenha a informação desejada.

3.4 CADS - *Confidential & Available* DEPSKY

O ADS garante a integridade e disponibilidade dos dados em *clouds* de armazenamento. No entanto, um dos problemas fundamentais neste tipo de solução é evitar que entidades não autorizadas tenham acesso aos dados armazenados na *cloud*.

Esta secção apresenta o algoritmo CADS, que integra um algoritmo criptográfico de *partilha de segredos* de tal forma que os dados armazenados em cada *cloud* individualmente sejam de pouca utilidade para um terceiro que intercepte ou obtenha a informação.

Um esquema de partilha de segredos [16] é o método para dividir um segredo entre um grupo de n participantes, em que a cada um deles é atribuída uma parte do segredo (que tem o mesmo tamanho do segredo original). O segredo pode ser reconstruído apenas quando $f+1$ dessas partes são recombinadas e qualquer combinação de até f partes individuais não revelam nenhuma informação sobre o segredo.

A diferença fundamental entre os protocolos de escrita do ADS e do CADS é que neste último introduzimos o algoritmo de partilha de segredos no passo 2 do ADS, de tal forma a produzir tantas partes do segredo (valor a ser escrito na unidade de dados) quanto o número de *clouds*. Cada uma destas partes é depois enviada para a sua respectiva *cloud*.

O algoritmo de leitura do CADS funciona de forma bastante similar ao ADS, porém, ao invés de aguardar apenas uma resposta com a versão mais actual dos dados (ADS - passo 2), esperam-se $f + 1$ partes de diferentes *clouds* para combiná-las usando o algoritmo de partilha de segredos, obtendo assim o valor originalmente escrito.

4 Concretização

O DepSky e todos os seus componentes foram concretizados na linguagem de programação Java. Em primeiro lugar foram concretizados alguns controladores, que são responsáveis pela comunicação com os diferentes sistemas de armazenamento em *clouds*. Cada controlador comunica com a respectiva *cloud* através de seus serviços web disponibilizados, utilizando uma interface REST. Toda a comunicação é efectuada sobre HTTP (ADS) ou HTTPS (CADS[2]). Os controladores foram os componentes que mais tempo consumiram em termos de concretização dada a variedade no funcionamento dos diferentes serviços web de cada *cloud*.

Foram concretizados controladores (com seus respectivos números de linhas de código) para os seguintes serviços: Amazon Simple Storage Service (175 LOC), Microsoft Windows Azure Platform (200 LOC), Nirvanix Storage Delivery Network (280 LOC),

[2] Requer canais confidenciais para garantir que as partes do segredo gerado são obtidas apenas pelas *clouds* a que se destinam.

DivShare (350 LOC), DocStoc (350 LOC) e Box.net (380 LOC). Os controladores do Amazon S3 e do Windows Azure foram concretizados sobre bibliotecas fornecidas pelos fornecedores do serviço ligeiramente modificadas para suportarem *proxies*. Após a conclusão de um número suficiente de controladores iniciou-se o desenvolvimento do componente responsável pelos protocolos (600 LOC), e de outro responsável pela verificação, validação e criação de metadados do sistema (250 LOC). Foi também concretizado um *wrapper* para controladores que efectua a gestão dos *retries* e *timeouts* dos pedidos HTTP (150 LOC), para garantir fiabilidade fim-a-fim. Finalmente, utilizou-se a biblioteca JSS (*Java Secret Sharing*) [5] para concretizar o esquema de partilha de segredos.

5 Avaliação

Nesta secção apresentamos uma avaliação do DepSky que tenta responder a três perguntas: Qual o custo adicional da utilização de replicação em *clouds* de armazenamento? Qual o ganho de desempenho e de disponibilidade na utilização de *clouds* replicadas para armazenar dados? Qual o custo relativo das diferentes versões (ADS, ADS com leitura optimizada e CADS) do DEPSKY?

5.1 Custo do Armazenamento Replicado

As *clouds* de armazenamento usualmente cobram pela quantidade de dados que entram, saem e ficam armazenados nos seus *data centers*. A tabela 1 apresenta os custos da utilização do modelo de *unidade de dados* apresentado neste artigo em diversas configurações do DEPSKY e em diferentes *clouds* individualmente[3]. A tabela mostra o custo em USD da realização de 10.000 operações de leitura e escrita para diferentes tamanhos de blocos de dados.

Operação	Tamanho	DEPSKY	DEPSKY opt. (melhor caso)	Amazon S3 (EU)	RackSpace	Windows Azure	Nirvanix
10k Leituras	100 kb	0,69	0,12	0,11	0,22	0,16	0,18
	1 Mb	6,54	1,02	1,01	2,20	1,51	1,80
	10 Mb	65,04	10,02	10,01	22,0	15,01	18,0
10k Escritas	100 kb	1,10	1,10	0,20	0,28	0,11	0,18
	1 Mb	4,84	4,84	1,10	0,80	1,01	1,80
	10 Mb	46,24	46,24	10,10	8,00	10,01	18,0

Tabela 1. Custo estimado, em USD, de 10.000 operações de leitura e escrita de dados com 100KB, 1MB e 10MB. É de salientar que os protocolos de leitura do DEPSKY efectuam 2 pedidos de leitura a cada *cloud*, e também, que os protocolos de escrita efectuam um pedido de leitura e 2 pedidos de escrita a cada *cloud*.

A coluna "DEPSKY" apresenta os custos do uso dos protocolos ADS e CADS propostos no artigo. É importante referir que o uso de confidencialidade (protocolo CADS)

[3] Nesta tabela apresentam-se os custos do RackSpace ao invés do Divshare (usado nas experiências de latência) devido ao facto do primeiro cobrar por uso, enquanto o segundo oferece apenas pacotes fixos.

não apresenta acréscimo representativo de custo uma vez que os seus metadados ocupam 500 bytes enquanto os metadados para unidades de dados sem confidencialidade ocupam cerca de 250 bytes.
A coluna "DEPSKY opt. (melhor caso)" apresenta os custos quando a optimização de leitura para o protocolo ADS é empregue. Neste caso, a política de escolha da *cloud* de leitura tem em conta não a que retornou os metadados mais rapidamente mas sim a que apresenta menor custo de leitura.

Custo de leitura. Este custo corresponde apenas ao custo de se ler os metadados da unidade de dados e os dados propriamente ditos. O custo de leitura do DEPSKY é similar à soma dos custos de leitura em cada uma das quatro *clouds* individualmente, enquanto que na versão optimizada temos o custo similar à Amazon S3, com um acréscimo de poucos cêntimos devido ao acesso dos metadados em todas as *clouds*.

Custo de escrita. O custo da escrita considera o custo de se ler os metadados, escrever uma nova versão destes e escrever a nova versão dos dados. Além disso, neste custo incluímos os custos de armazenamento dos dados, o que significa que estamos a considerar um sistema onde nenhuma versão de uma unidade de dados será apagada (i.e., escritas apenas criam novas versões). Conforme já referido, esta funcionalidade é importante para dados críticos na medida em que permite recuperar versões anteriores das unidades de dados armazenadas. No entanto, na prática esse custo pode ser amortizado apagando-se versões antigas.

Os resultados da tabela mostram que o custo apresentado para as versões do DEPSKY correspondem à soma dos custos de escrita nas *clouds*. Estes custos, assim como na leitura não-optimizada, advêem do modelo de replicação utilizado onde armazenamos o bloco de dados em todas as *clouds*. Se aplicássemos técnicas similares ao RAID nível 5 [15], esses custos cairiam pela metade, já que os dados armazenados em cada *cloud* teriam aproximadamente metade do tamanho da unidade de dados.

5.2 Desempenho e Disponibilidade

O DEPSKY foi desenhado tendo em conta cargas de trabalho em que leituras são muito mais frequentes que escritas, como é observado em praticamente todos os sistemas de armazenamento [10]. Assim, a nossa avaliação concentrou-se na latência das operações de leitura em diferentes configurações. No entanto, são reportados alguns valores de latência do protocolo de escrita no fim desta secção para fins de completude do estudo.

Metodologia. As medidas de latência de leitura foram obtidas através de uma aplicação que acede aos dados de 7 formas diferentes (configurações): às 4 *clouds* de armazenamento individualmente (Amazon S3, Windows Azure, Nirvanix e Divshare) e às 3 versões do protocolo de leitura do DEPSKY (ADS, ADS com leituras optimizadas e CADS). Todas as versões do DEPSKY usam as quatro *clouds* mencionadas para armazenar dados, e portanto toleram uma falha.

Foram medidos tempos de leitura para unidades de dados de três tamanhos: 100K, 1M e 10M bytes. A aplicação executou todas estas leituras periodicamente - de um em um minuto (10K e 1M) ou de cinco em cinco minutos (10M) - e armazenou os

tempos obtidos em ficheiros de log. O objectivo foi ler os dados através das diferentes configurações num intervalo de tempo o mais curto possível tentando minimizar as variações de desempenho da Internet.

As experiências foram realizadas entre 31 de Maio e 14 de Junho de 2010, com o cliente a executar numa máquina do Departamento de Informática da FCUL e a aceder às quatro *clouds* de armazenamento. Foram efectuadas 99.414 medidas de latência, correspondento a 14.202 medidas por cada uma das 7 configurações.

Em todos os testes do DEPSKY, o custo dos algoritmos de criptografia (assinatura, verificação e partilha de segredos) foi inferior a 20 ms, o que corresponde a menos de 2% da menor latência observada nos protocolos. Isto era esperado uma vez que a latência de comunicação da Internet e o processamento adicional para acesso de serviços web tende a dominar o tempo de execução de qualquer aplicação neste ambiente.

Latência de leitura. A figura 2 apresenta a função de distribuição cumulativa das latências medidas na leitura dos diversos tamanhos dos dados nas diversas *clouds* individualmente e utilizando as diferentes versões do DEPSKY. São apresentados resultados relativos ao acesso às *clouds* individualmente (figuras 2(a), 2(c) e 2(e)) e utilizando diferentes versões do DEPSKY (com os resultados da Amazon S3, para fins de comparação - figuras 2(b), 2(d) e 2(f)).

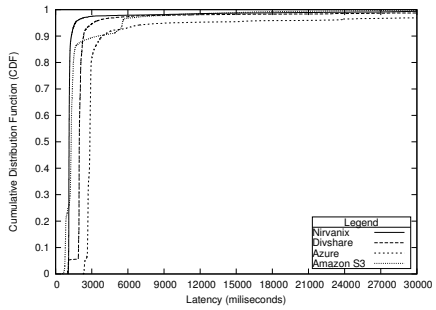
Nas unidades de dados de 100K podemos observar que as distribuições de latências para todas as configurações são bastante semelhantes. Já nas experiências com unidades de dados de 1M, podemos observar alguma discrepância entre os desempenhos da Amazon S3 e da Nirvanix. Além disso, com este tamanho de blocos já se percebe a diferença entre o uso de replicação de dados em *clouds* e uma única *cloud*: 90% das leituras optimizadas com o DEPSKY estão abaixo de 3,2 segundos, enquanto na S3 este valor aproxima-se dos 8 segundos.

As experiências com unidades de dados de 10M já demonstram as dificuldades em lidar-se com o armazenamento de dados na Internet. Na figura 2(e) observa-se uma larga discrepância entre os resultados observados para a Nirvanix e a Divshare quando comparados com os resultados da Azure e da S3. Em particular, no decorrer destas experiências observou-se um largo período de indisponibilidade da Azure (ver a seguir), o que é representado no gráfico pelos 20% dos resultados de latência que não aparecem na figura.

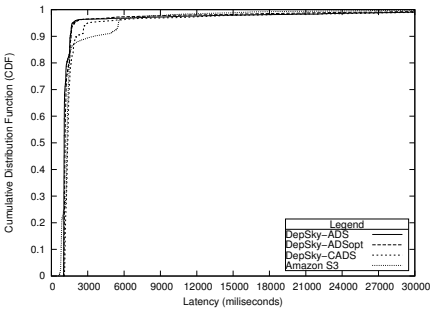
No que diz respeito às diversas versões do DEPSKY, a figura 2(f) mostra que neste caso a replicação dos dados em diversas *clouds* diminui de forma significativa a latência de leitura, mesmo na versão optimizada em que não se tenta ler de várias *clouds* mas apenas da que retornou os metadados mais rapidamente. De notar também que o uso da primitiva de partilha de segredos para confidencialidade (protocolo CADS), torna o DEPSKY muito menos eficiente já que é necessário obter os dados de duas *clouds* diferentes, ao invés de uma (como acontece nas outras versões).

Falhas. Durante as experiências foram observadas várias falhas no acesso aos sistemas de armazenamento, conforme reportado na tabela 2.

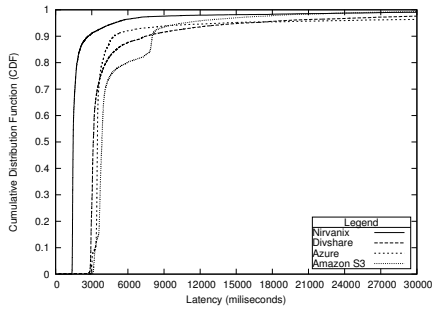
Durante os testes observámos um período de instabilidade e indisponibilidade na *cloud* Azure entre as 11h e as 21h do dia 10 de Junho (GMT+1, fuso horário de verão de Portugal continental). Neste período mais de 95% dos pedidos de leitura dos dados foram rejeitados com a mensagem de erro *"Unable to read complete data from server"*.



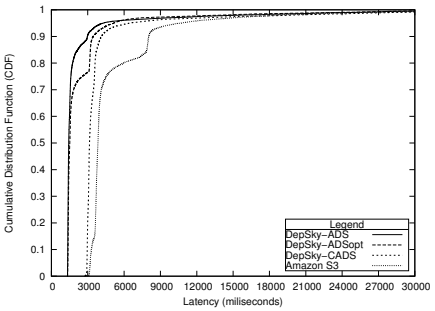
(a) Clouds - Unidades de dados de 100K.



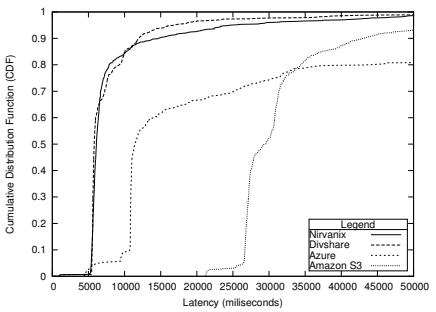
(b) DEPSKY - Unidades de dados de 100K.



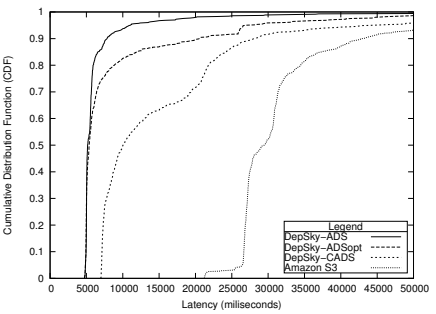
(c) Clouds - Unidades de dados de 1M.



(d) DEPSKY - Unidades de dados de 1M.



(e) Clouds - Unidades de dados de 10M.



(f) DEPSKY - Unidades de dados de 10M.

Figura 2. Função de distribuição cumulativa para as latências de leitura observadas em quatro diferentes *clouds* (Amazon S3, Windows Azure, Nirvanix e DivShare) e nas três versões do DEPSKY replicando dados nas mesmas *clouds*.

Experiência	Todas	Amazon S3	Azure	Nirvanix
100K	1	0	0	1
1M	1	9	13	2
10M	0	0	10+10hs	7

Tabela 2. Número de falhas observadas durante as experiências de leitura. O "10+10hs" para a Azure na unidade de dados de 10M significa que para além das 10 falhas reportadas houve um período de 10 horas onde mais de 95% dos acessos individuais a este sistema falharam.

Para além deste evento, o número de operações mal sucedidas é bastante pequeno se tivermos em conta a quantidade total de operações executadas. É de salientar que o DEPSKY falhou apenas duas vezes, quando todas as *clouds* estavam indisponíveis. Estas falhas aconteceram possivelmente devido a problemas de conectividade na saída da rede do DI/FCUL.

Latência de escrita. Para fins de completude da nossa avaliação reportamos na tabela 3 os tempos médios de escrita para diferentes configurações (obtidos a partir de um conjunto de 1000 escritas para cada tamanho de unidade de dados, executadas no dia 14 de Junho).

Dados	DEPSKY-ADS	DEPSKY-CADS	Amazon S3	DivShare	Azure	Nirvanix
100K	1767	2790	2336	3287	2801	2802
1M	4376	4928	5640	4684	3823	2626
10M	20315	24012	32204	8298	20017	4816

Tabela 3. Latência média (ms) de escrita para diferentes tamanhos de unidades de dados, configurações do DEPSKY e *clouds* de armazenamento.

Estes resultados mostram que algumas *clouds* (Divshare e Nirvanix) apresentam pouco aumento de latência quando passamos a escrever altos volumes de dados, enquanto outras (Azure e S3), apresentam uma perda de desempenho proporcional ao tamanho dos dados a serem escritos. O desempenho das versões do DEPSKY é similar a estas versões mais lentas na medida que os protocolos de escrita requerem a confirmação de escrita em 3 das 4 *clouds* utilizadas.

6 Trabalhos Relacionados

Até onde sabemos, existem apenas dois trabalhos bastante recentes que tentam fazer algo similar ao DEPSKY para melhorar a confiabilidade e segurança dos dados armazenados nas *clouds*, tendo sido ambos desenvolvidos em paralelo com o trabalho aqui reportado.

O HAIL (*High-Availability Integrity Layer*) [9] consiste num conjunto de protocolos criptográficos que juntam códigos de apagamento com provas de recuperação que permitem a concretização de uma camada de software para proteger a integridade dos dados armazenados em *clouds*, mesmo que estas sejam invadidas e corrompidas por um adversário móvel. Quando comparado ao DEPSKY, o HAIL apresenta pelo menos três limitações: só lida com dados estáticos (i.e., os algoritmos não suportam actualizações e múltiplas versões dos dados), requer que os servidores executem código (ao contrário do DEPSKY, que considera as *clouds* de armazenamento como discos passivos) e não usa nenhum mecanismo para protecção da confidencialidade dos dados armazenados.

O sistema RACS (*Redundant Array of Cloud Storage*) [11] utiliza técnicas similares às utilizadas nos sistemas RAID nível 5 [15] para concretizar replicação de dados em diversas *clouds*. Diferentemente do DEPSKY, o RACS não se preocupa com problemas de segurança, mas sim com possíveis "falhas económicas", onde uma *cloud* aumenta o custo do seu serviço de tal forma que torna inviável o acesso aos dados. Além de

não proteger contra corrupção de dados e violações de confidencialidade, o RACS também não suporta actualizações dos dados armazenados. Todas estas limitações tornam o RACS menos abrangente do que o DEPSKY.

Além das diferenças entre os sistemas, os trabalhos sobre o HAIL e RACS não apresentam nenhum tipo de medida que utiliza diversidade de *clouds*.

7 Conclusão

Neste trabalho foi apresentado o DEPSKY, um sistema que fornece disponibilidade, integridade e confidencialidade de informação armazenada na *cloud*. Na avaliação experimental demonstrou-se que o DEPSKY não fica muito aquém, em termos de desempenho, dos serviços testados. Podemos afirmar que com o DEPSKY temos sempre o melhor serviço independentemente das condições de cada *cloud*.

Os trabalhos actuais e futuros deverão concentrar-se na inclusão de códigos de apagamento para diminuir o tamanho dos blocos de dados armazenados (de forma similar ao RAID [15]) e numa avaliação da disponibilidade e desempenho das *clouds* a partir de diferentes localizações na Internet.

Agradecimentos. Este trabalho foi suportado pela FCT através de seu programa multianual (LaSIGE) e do projecto CloudFIT (PTDC/EIA-CCO/108299/2008).

Referências

1. Amazon Simple Storage Service (`https://s3.amazonaws.com`), June 2010.
2. Box.net (`http://www.box.net`), June 2010.
3. Divshare (`http://www.divshare.com`), June 2010.
4. Docstoc (`http://www.docstoc.com`), June 2010.
5. Java secret sharing (`http://www.navigators.di.fc.ul.pt/software/jitt/jss.html`), June 2010.
6. Microsoft Windows Azure Platform (`http://www.windowsazure.com`), June 2010.
7. Nirvanix Storage Delivery Network (`http://www.nirvanix.com`), June 2010.
8. Rackspace (`http://www.rackspace.com`), June 2010.
9. Kevin D. Bowers, Ari Juels, and Alina Oprea. HAIL: a high-availability and integrity layer for cloud storage. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198, New York, NY, USA, 2009. ACM.
10. Gregory Chockler, Rachid Guerraoui, Idit Keidar, and Marko Vukolić. Reliable Distributed Storage. *IEEE Computer*, 42(4):60–67, 2009.
11. L. Princehouse H. Abu-Libdeh and H. Weatherspoon. RACS: A case for cloud storage diversity. *ACM Symposium on Cloud Computing (SOCC)*, June 2010.
12. Leslie Lamport. On Interprocess Communication. Part I: Basic Formalism. *Distributed Computing*, 1(2):77–85, 1986.
13. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
14. Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, October 1998.
15. David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116, New York, NY, USA, 1988. ACM.
16. Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

On using Constraints for Network Intrusion Detection

Pedro Salgueiro and Salvador Abreu

Departamento de Informática, Universidade de Évora
and CENTRIA FCT/UNL, Portugal
{pds,spa}@di.uevora.pt

Abstract. In this work we present a domain specific language for network intrusion detection that allows to describe network intrusions composed by several network packets, using a declarative approach to describe the desirable network situations, and based on that description, a set of parameterizations for network intrusion detection mechanisms based on Constraint Programming(CP) will execute to find those intrusions.

Keywords: Constraint Programming, Intrusion Detection Systems, Domain Specific Languages

Resumo Neste trabalho apresentamos uma linguagem específica de domínio para descrever intrusões de rede, capaz de descrever ataques compostos por vários pacotes de rede, e a partir de uma descrição declarativa da intrusão, criar um conjunto de ferramentas capaz de detectar a intrusão desejada, tendo como base a Programação por Restrições(CP).

Palavras Chave: Programação por restrições, Sistemas de detecção de Intrusões, Linguagens Específicas de Domínio

1 Introduction

Computer networks are composed of numerous complex and heterogeneous systems, hosts and services. Maintaining the security of the networks is crucial to keep the users data safe, which may be accomplished by an Intrusion Detection System (IDS) e.g. `Snort` [1,2]

To maintain the quality and integrity of the services provided by a computer network, some aspects must be verified in order to maintain the security of the users data. The description of those conditions, together with a verification that they are met can be seen as an Intrusion Detection task. These conditions, specified in terms of properties of parts of the (observed) network traffic, will amount to a specification of a desired or an unwanted state of the network, such as that brought about by a system intrusion or another form of malicious access.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 637–648

Those conditions can naturally be described using a declarative programming approach, such as Constraint Programming [3] or Constraint Based Local Search Programming (CBLS) [4], enabling the description of these situations in a declarative and expressive way. To help the description of those network situations, we created NeMODe, a Domain Specific Language (DSL) [5], which enables an easy description of intrusion signatures that spread across several network packets, which will then translate the *program* into constraints that will be solved by more than one constraint solving techniques, including Constraint Based Local Search and Propagation-based systems such as Gecode [6]. It will also have the capabilities of running several solvers in parallel, in order to benefit from the earliest possible solution.

Throughout this paper, we mention technical terms pertaining to TCP/IP and UDP/IP network packets, such as *packet flags, ACK, SYN, RST, acknowledgment, source port, destination port, source address, destination address, payload*, which are described in [7].

1.1 Intrusion Detection Systems

Intrusion Detection Systems(IDS) play an important role in computer network security, which focus on traffic monitoring trying to inspect traffic to look for anomalies or undesirable communications in order to keep the network a safe place. There are two major methods to detect intrusions in computer networks; 1) based on the network intrusion signatures, and 2) based on the detection of anomalies on the network [8]. In this work, we adopted an approach based on signatures.

Snort is a widely used Intrusion Detection System that relies on pattern-matching techniques to detect the network attacks [9]. Snort is a very efficient Intrusion Detection System but is primarily designed to detect network attacks which have a signature that can be identified in a single network packet. Although it provides some basic mechanisms to write rules that spread across several network packets, the relations between those network packets are very simple and limited, such as the `Stream4` and `Flow` pre-processor.

Most of the recent work in intrusion detection systems has been focused on the performance [10], but there has been also some work [10,11] that focus on the method used to match the network packet signatures and the type of signatures that can be detected, using alternative search methods that allows the search of signatures that spreads across several packets, which is one of the limitations of Snort and most other intrusion detection systems.

1.2 Constraint Programming

Constraint Programming (CP) is a declarative programming paradigm which consists in the formulation of a solution to a problem as a *Constraint Satisfaction Problem* (CSP) [3], in which a number of variables are introduced, with well-specified domains and which describe the state of the system. A set of relations, called *constraints*, is then imposed on the variables which make up the problem. These constraints are understood to have to hold true for a particular set of bindings for the variables, resulting in a *solution* to the CSP.

Adaptive Search Adaptive Search (AS) [12] is a Constraint Based Local Search [4] algorithm, taking into account the structure of the problem and using variable-based information to design general heuristics which help solve the problem. AS has an adaptive memory similar to TabuSearch [13] in order to prevent stagnation while solving the problem. The solving method of AS relies on iterative repairs based on variable and constraint error information that seeks to reduce errors on the used variables, in order to reach a valid solution, i.e. a solution with error value of zero. Adaptive Search has recently been ported to Cell/BE, presented in [14].

Gecode Gecode [15] is a constraint solver library implemented in C++, designed to be interfaced with other systems or programming languages. Gecode is a Propagation-based [3] constraint solver system, involving variables ranging over some finite set of integers to solve the problems, being described by stating constraints over each variable of the problem, which states the allowed values for each variable , then, the constraint solver propagates all the constraints and reduce the domain of each network variables in order to satisfy all the constraints and instantiate the variables with valid values, thus reaching a solution to the initial problem.

2 Network Monitoring with Constraints

Our approach to intrusion detection relies on being able to describe the desired signatures through the use of constraints and then identify the set of packets that match the target network situation in the network traffic window, which is a log of the network traffic in a given time interval.

The problem needs to be modeled as a Constraint Satisfaction Problem (CSP) in order to use the constraint programming mechanisms. Such a CSP will be composed by a set of variables, V, representing the network packets, their associated domains, D, and a set of constraints, C that relates the variables in order to describe the network situation. On such a CSP, each network packet variable is a tuple of integer variables, 19 for TCP/IP [1] packets and 12 for UDP packets [2], which represent the significant fields of a network packet necessary to model the intrusion signatures that we have used in our experiments, such as the timestamp, source/destination addresses, source/destination ports and packet data, which are used in both TCP and UDP packets, the extra TCP flags and packet sequence number fields are used with TCP packets. The number of fields may increase over time with the evolution of the work and the use of more complex intrusions. The domain of the network packet variables, D, are the values actually seen on the network traffic window, which is a set of tuples of 19 integer values, each tuple representing a network packet actually observed on the traffic

[1] Here, we are only considering the "interesting" fields in TCP/IP packets, from an IDS point-of-view.

[2] Here, we are only considering the "interesting" fields in UDP packets, from an IDS point-of-view.

window and each integer value represents the fields that are relevant to network monitoring. The packets payload is stored separately in an array containing the payload of all packets seen on the traffic window. A solution to such a CSP, if it exists, is the set of packets that correspond to the network intrusion described by the CSP. Listing 1 shows a representation of such CSP, where P represents a set of network packet variables, D, a set of network packets representing the network traffic window, DP the payloads of all packets and $\forall P_i \in P \Rightarrow P_i \in D$ the associated domains of each variable. The payload of packet i would be $DP[i]$. Performing Network Monitoring with constraints is explained in more detail on the work presented in [16].

Listing 1 Representation of a network CSP

$$P = \{(P_{1,1}, \ldots, P_{1,19}), \ldots, (P_{n,1}, \ldots, P_{n,19})\}$$
$$D = \{(V_{1,1}, \ldots, V_{1,19}), \ldots, (V_{m,1}, \ldots, V_{m,19})\}$$
$$DP = \{DP_1, \ldots, DP_m\}$$
$$\forall P_i \in P \Rightarrow P_i \in D$$

3 NeMODe - A DSL to describe network signatures

In this work we present a simple, declarative, intuitive domain-specific programming language for the Network Intrusion Detection [5] domain called NeMODe. NeMODe language talks about network entities, network entity properties and relations between network entities and network entity properties, which allows to describe network intrusion signatures, and with base on those description, generate Intrusion Detection mechanisms. A more complete description of this DSL as well as other examples are presented in [16], which is an extended description of the work described in this paper.

The key characteristic of NeMODe is to ease the way how network attack signatures are described using constraint programming, hiding from the user all the constraint programing aspects and complexity of modeling network signatures in a Constraint Satisfaction Problem(CSP), but still using the methodologies of CP to describe the problem at a much higher level, describing how the network entities should relate among each other and what properties they should verify. Maintaining the declaritivity and expressiveness of the CP allows an easy and intuitive way of describing the network attack signatures, by describing the properties that must or must not be seen on the individual network packets, as well as the relationships that should or should not exist between each of the network packets.

NeMODe is a front-end to several back-ends, one to each intrusion detection mechanism, allowing to generate several detection mechanisms from a single description. Having a single specification to several constraint solvers allows the

search of a solution using different methods of search, allowing to run each of those methods in parallel, which allows to obtain different results from each solver. Depending on the characteristics of the problem, some solvers could produce a better and faster solution that others, allowing to choose the first solution to be produced.

NeMODe presents five groups of *statements*: (1) the primitives of the language, (2) the connectives, (3) definitions, (4) the use of such definitions and (5) macro statements. The primitives are the basic statements of the language, which state simple properties that each network variable should verify. The connectives are statements that relate two or more network variables, forcing them to verify some relations. The definition is a simple way of storing primitives or connectives under a variable to be used later. The *use* of definitions, forces a previous definition to used. Finally, the macro statements, are helpers that avoid unnecessary code repetition and ease the description of the signature.

The following list presents the set of primitives (*predicates*) available in the current implementation of NeMODe which allows to state properties of network packets that should be verified.

- Force a variable to be a network packet.
- Force a variable to be a TCP or UDP packet.
- Force a packet to have specific a TCP flag set.
- Force a packet not to acknowledge any packet.
- Force a packet to contain a string.

Follows a list of the *connective* statements, which are used to relate several network entities.

- Force a packet to acknowledge other packet.
- Restrict the temporal distance between packets.
- Force two packets to be related.
- Force the source/destination port of a packet.
- Force the source/destination address of a packet.
- Force two packet to contain the same piece of data in a given position and size on their payload.

NeMODe provides a special type of statements to help users specify network signatures with minimum work, the `definition` statements. These statements allows to store a set of properties over a set of network entities and give it a name and using them later on the program. Listing 2 shows an example of a simple `definition` where some properties over two network packets are stated, in this particular case, the variable `A` should be a TCP/IP packet, and have its `syn` flag set. These set of properties are *stored* in variable `C`, which can later be used. Those definitions by them self don't have any effect, they are only applied when used or referred. In order to use those definitions, simply refer the variable to which the set of properties was assigned or use it in a `macro` statement, explained next.

The `macro` statements provide mechanisms to help the user describe the situation, by avoiding unnecessary code repetition. This `macro` statements can

Listing 2 Example of a definition

```
1: C = { packet(A),
2:       tcp(A), syn(A) }
```

Listing 3 Example of `macro function`

```
1:     C = { packet(A), syn(A) },
2:     R:=repeat(3,C),
3:     max_duration(R) < secs(60)
```

be used to repeat a set of properties assigned to a variable, and give a name to that repetition, allowing future references to each property of each instance of the repetition i.e. `R:=repeat(3,C)`. Other type of `macro` statements are the ones that are applied to the repetitions *stored* in a variable, such as state the maximum or minimum allowable time interval between each instance of the repetition, i.e. `max_duration(R) < secs(60)` or the maximum/minimum overall interval time that a repetition can take, i.e. `max_interval(R) < secs(60)`. Listing 3 illustrates a simple use of this macro functions.

Listing 4 Accessing a variable

```
1:     C = { packet(A), syn(A)},
2:     R := repeat(3,C),
3:     nak(R[1]:A)
```

When using the `repeat` statement, as in line 2 of Listing 4, each instance of the repetition as well as its variables keeps accessible, referring it as the *nth* instance and then referring the variables name, i.e. `R[1]:A`. Listing 4 shows an example, where the statement `nak` is applied to variable `A` of the first instance of the repetition `R`.

3.1 Available back-ends

NeMODe provides two back-end detection mechanisms; (1) based on the Gecode constraint solver and (2) based on the Adaptive Search algorithm.

Each of these detection mechanisms are based on Constraint Programming techniques, but they are completely different in the way they perform the detection, and also the way the signatures are described. In Sec. 1.2 each of these approaches are explained.

3.2 Examples

So far, we have worked with some simple network intrusion signatures: (1) a DHCP spoofing, (2) a DNS spoofing and (3) a SYN flood attack. All of these intrusion patterns can be described using NeMODe and the generated code was

successful in finding the desired situations in the network traffic logs. A Portscan attack and an SSH Password brute-force attack are further explained in [16].

DHCP spoofing DHCP Spoofing is a Man in The Middle(MITM) attack, where the attacker tries to reply to a DHCP request faster than the legit DHCP server of the local network, allowing the attacker to provide false network configurations to the target host, such as the default gateway, forcing all traffic from/to the target to pass though an attacker controlled machine, allowing it to capture or modify the important data. This kind of intrusion can be detected by looking for several answers to a single DHCP request, originated in different machines. If this situation is found in the traffic of some network, there is a great probability that someone is performing this kind of attack. A NeMODe program to model a DHCP spoofing is shown in Listing 5. Lines 2 and 3 describes the packet that initiates a requests a DHCP, lines 5 and 6 the first reply to the request and lines 8 and 9 the second reply the DHCP request. Finally, on line 11 is stated that packets `B` and `C`(the first and second reply) should have different source addresses.

Listing 5 A DHCP Spoofing attack programmed in NeMODe

```
1:  dhcp_spoofing {
2:      packet(A), udp(A),
3:      dst_port(A)==67,
4:
5:      packet(B), udp(B),
6:      dst_port(B)==68,
7:
8:      packet(C), udp(C),
9       dst_port(C)==68,
10:
11:     src(B) != src(C)
12: } => {
13:     alert('DHCP Spoofing attempt')
14: };
```

DNS spoofing DNS Spoofing is also a Man in The Middle (MITM) attack. In this attack, the attacker tries to provide a false DNS query posted by the victim, if succeeded the victim could access a machine under the control of the attacker, thinking that it is accessing the legit machine, allowing the attacker to obtain crucial data from the victim. In order to arrange this attack, the attacker tries to respond with a false DNS query faster than the legit DNS server, providing a false IP address to the name that the victim was looking for. This kind of attacks is possible to detect by looking for several replies to the same DNS query. Listing 6 shows how this attack can be programmed using NeMODe. Line 2 and 3 describes the packet that makes the DNS request. Lines 5-8, describes a first reply to the DNS request and lines 10-13 describes the second reply. Lines 15-17 states that packets `B` and `C` should be different and that the *DNS id* of the

replies should be the equal to the DNS request, which is the first two bytes of the packets data.

Listing 6 A DNS Spoofing attack programmed in NeMODe

```
1: dns_spoofing {
2:     packet(A), udp(A),
3:     dst_port(A)==53,
4:
5:     packet(B), udp(B),
6:     dst(B)==src(A),
7:     src_port(B)==53,
8:     dst_port(B)==src_port(A),
9:
10:    packet(C), udp(C),
11:    dst(C) == src(A),
12:    src_port(C) == 53,
13:    dst_port(C) == src_port(A),
14:
15:    B != C,
16:    data(B,0,2) == data(A,0,2),
17:    data(C,0,2) == data(A,0,2)
18:} => {
19:  alert('DNS Spoofing attempt')
20:};
```

SYN flood attack A SYN flood attack happens when the attacker initiates more TCP/IP connections than the server can handle and then ignoring the replies from the server, forcing the server to have a large number of half open connections in standby, which leads the service to stop when this number reach the limit of number of connections. This attack can be detected if a large number of connections is made from a single machine to other in a very short time interval. Listing 7 shows how a SYN flood attack can be described using NeMODe. Line 2-5 describes a TCP/IP packet with the SYN flag and assigns those properties to variable `C`. In line 7, the *macro* statement *repeat* is used to repeat the properties of definition `C` 30 times, and assign it to variable `R`. Line 8 states that the time interval between each repetition of C should be less than to 500 micro-seconds.

Listing 7 A SYN flood attack programmed with NeMODe

```
1:  syn_flood {
2:    C = {
3:           packet(A), tcp(A),
4:           syn(A), nak(A)
5:    },
6:
7:    R := repeat(30,C),
8:    max_interval(R) < usecs(500)
9: } => {
10:   alert('SYN flood attack attempt')
11: };
```

3.3 Code Generation

The current implementation of NeMODe is able to generate code for the Gecode solver and for the Adaptive Search algorithm. These two approaches to constraint solving are completely different as well as the description of the problems, forcing us to have several code generators for each of back-end available. We were able to minimize this difference by creating custom libraries for each constraint solver so that the code generation process is not completely different for each back-end.

Generating an A.S. program The task of generating Adaptive Search resumes to create the proper error functions so that Adaptive Search be able to solve the problem; the `cost_of_solution` and `cost_on_variable`. To ease the generation of this functions, a small library was created which implements small error functions, specific to the network intrusion detection domain, which are then used to generate the code for the error functions.

Generating a Gecode program This goal is achieved by generating code based on Gecode constraint propagators that describe the desired network signatures. We created a custom library that defines functions that combine several stock Gecode constraints to define custom, network related "macro" constraints. The same library includes definitions for a few network-related constraint propagators, useful to implement some of the constraints needed to describe and solve IDS problems.

4 Experimental Results

While developing this work, several experiments were done. We have tested the examples of Sect. 3.2, a DHCP Spoofing attack, a DNS Spoofing attack and a SYN flood attack. All these network intrusions were successfully described using NeMODe and valid Gecode and Adaptive Search code were produced for all network signatures. The code generated by NeMODe was then executed in order to validate the code and ensure that it could indeed find the desired network intrusions.

The code generated for Gecode was run on a dedicated computer, an HP Proliant DL380 G4 with two Intel(R) Xeon(TM) CPU 3.40GHz and with 4 GB of memory, running Debian GNU/Linux 4.0 with Linux kernel version 2.6.18-5. As for the Adaptive Search code, it run on an IBM BladeCenter H equipped with QS21 dual-Cell/BE blades, each with two 3.2 GHz processors, 2GB of RAM, running RHEL Server release 5.2.

The reason to run both detection mechanisms in different machines with a completely different architecture is because Adaptive Search has recently been ported to Cell/BE, and we choose this version of Adaptive Search to run our experiments, forcing us to use the QS21 dual-Cell/BE blades, which is incompatible with the implementation of Gecode, forcing us to use a machine with x86 architecture to run Gecode.

Table 1. Average time(in seconds) necessary to detect the intrusions using Gecode and Adaptive Search

Intrusion to detect	Gecode (seconds)	A.S (seconds)
DHCP Spoofing	0.0082	0.3924
DNS Spoofing	0.0069	0.3512
SYN flood	0.0566	0.0466

In all the experiments we used log files representing network traffic which contains the desired signatures to be detected. These log files were created with the help of `tcpdump` [17], which is a packet sniffer, during an actual attack to a computer, which was induced to simulate the real attacks described in this work.

DHCP spoofing and DNS spoofing attacks For the DHCP spoofing and DNS spoofing attack, we used tcpdump to capture a log file, composed of 400 network packets, while a computer was under an actual attack. We used Ettercap to perform the DHCP spoofing and DNS spoofing attacks. The attack was programmed in NeMODe, which successfully generated code for Adaptive Search as well as for Gecode and successfully detected the intrusions.

SYN flood attack In the SYN flood attack a log file of 100 network packets was created with the help of tcpdump while a computer was under a SYN flood attack. The attack was programmed in NeMODe which in turn generated code for Adaptive Search and Gecode. This code was then used to successfully detect the intrusion.

4.1 Results

Table 1 presents the time(user time, in seconds) required to find the desired network situation for each of the attacks presented in the present work, using both detection mechanisms, Gecode and Adaptive Search. The execution times presented in Table 1 are the average times of 128 runs.

5 Evaluation

The performance of the prototypes described in Sec. 4 shows a multitude of performance numbers relative to the intrusion detection mechanisms used for each network signature. Looking closely at the results in Table 1, it is possible to see that Gecode usually performs better than Adaptive Search, except in the SYN flood attack. This difference is explained by the fact that Adaptive Search needs a very good heuristic functions to improve its performance. We created some heuristics based on the network situations we are studying which improved the performance of Adaptive Search, but still can't reach the performance of Gecode. The SYN flood attack performed better in Adaptive Search due to the

fact that the network packets of the attack are close together and there aren't almost any other packets between the packets of the attack.

Even without a perfect heuristic of Adaptive Search, the results obtained are quite encouraging. As for Gecode, the results obtained are quite good. With these results, we are now ready to start the detection of intrusions in real network traffic instead of log files.

As for NeMODe, it turns out to be a success, since it was possible to easily describe all the three network intrusions and generate valid code that could detect the desired network situation. Although other intrusion detection systems like Snort could detect the attacks presented in this work, they can not describe the problems with the expressiveness used by NeMODe or even relate the several packets that make part of the attack.

6 Conclusions and Future Work

The work presented in this paper presents NeMODe, a Domain Specific Language to describe network intrusion signatures that generates intrusion detection recognizers based on Constraint Programming, more specifically, using Gecode and Adaptive Search.

The results obtained in this work show that it's possible to transform the description of a network situation using several intrusion detection mechanisms, based on Constraint Programming, from a single description and then use those recognizers detect the desired intrusion using the generated code, demonstrating the viability of using Constraint Programming in network monitoring tasks.

Also, we showed that we can easily describe network signature attacks that spread across several network packets, which is somewhat tricky or even impossible to make using systems like Snort. Although the intrusion mentioned in this work can be detected with other intrusion detection systems, they are modeled/described with out relating the several network packets of the intrusion, much of the times using a single network packet to describe the intrusion, which could in some situations produce a large number of false positives.

This work is still at an early stage of development, we expect there to be plenty of room for improvement: we have reached an efficiency level that may be suitable to start performing network monitoring tasks on live network traffic link, meaning that and important step will be to apply this method in a real network to assess its performance.

A very important future work is to model more network situations as a CSP in order to evaluate the performance of the system while working with a larger diversity of problems.

NeMODe will be extended to be more flexible, allowing to describe other network properties and a broader range of attack signatures and also include more back-ends, allowing the detection of intrusions using several methods using a single description.

Acknowledgments

Pedro Salgueiro acknowledges FCT –Fundação para a Ciência e a Tecnologia– for supporting him with scholarship SFRH/BD/35581/2007. The IBM QS21 dual-Cell/BE blades used in this work were donated by IBM Corporation, in the context of a SUR (Shared University Research) grant awarded to Universidade de Évora and CENTRIA.

References

1. Martin Roesch. Snort - lightweight intrusion detection for networks. In *LISA '99: Proceedings of the 13th USENIX conference on System administration*, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.
2. Jay Beale. *Snort 2.1 Intrusion Detection, Second Edition*. Syngress Publishing, 2004.
3. F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier Science, 2006.
4. P. Van Hentenryck and L. Michel. *Constraint-based local search*. MIT Press, 2005.
5. A. Van Deursen and J. Visser. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36, 2000.
6. Gecode Team. Gecode: Generic constraint development environment, 2008. Available from http://www.gecode.org.
7. Douglas Comer. *Internetworking With TCP/IP Volume 1: Principles Protocols, and Architecture, 5th edition*. Prentice Hall, 2006.
8. Y. Zhang and W. Lee. Intrusion detection in wireless ad-hoc networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, page 283. ACM, 2000.
9. H. Song and J.W. Lockwood. Efficient packet classification for network intrusion detection using FPGA. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 238–245. ACM New York, NY, USA, 2005.
10. K.S.P. Arun. Flow-aware cross packet inspection using bloom filters for high speed data-path content matching. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 1230 –1234, 6-7 2009.
11. S. Kumar and E.H. Spafford. A software architecture to support misuse intrusion detection. In *Proceedings of the 18th national information security conference*, pages 194–204, 1995.
12. P. Codognet and D. Diaz. Yet another local search method for constraint solving. *Lecture Notes in Computer Science*, 2264:73–90, 2001.
13. F. Glover, M. Laguna, et al. *Tabu search*. Springer, 1997.
14. Salvador Abreu, Daniel Diaz, and Philippe Codognet. Parallel local search for solving constraint problems on the cell broadband engine (preliminary results). *CoRR*, abs/0910.1264, 2009.
15. C. Schulte and P.J. Stuckey. Speeding up constraint propagation. *Lecture Notes in Computer Science*, 3258:619–633, 2004.
16. Pedro Salgueiro and Salvador Abreu. A DSL for Intrusion Detection based on Constraint Programming. In *SIN 2010: Proceedings of the 3rd International Conference on Security of Information and Networks*, New York, NY, USA, 2010. ACM.
17. tcpdump web page at http://www.tcpdump.org/, April, 2009.

TYPHON: Um Serviço de Autenticação e Autorização Tolerante a Intrusões

João Sousa, Alysson Bessani, and Paulo Sousa

Laboratório de Sistemas Informáticos de Grande Escala,
Faculdade de Ciências da Universidade de Lisboa
Lisboa, Portugal

Resumo A norma Kerberos v5 especifica como é que clientes e serviços de um sistema distribuído podem autenticar-se mutuamente usando um serviço de autenticação centralizado. Se este serviço falhar, por paragem ou de forma arbitrária (e.g., bug de software, problema de hardware, intrusão), os clientes e serviços que dependem dele deixam de poder autenticar-se. Este artigo apresenta um serviço de autenticação e autorização que respeita a especificação do Kerberos v5 tal como é descrita no RFC 4120, fazendo uso da técnica da replicação da máquina de estados e de componentes seguros para tornar o serviço mais resiliente. A técnica da replicação da máquina de estados utilizada oferece tolerância a faltas arbitrárias, enquanto os componentes seguros garantem que as chaves dos clientes e dos serviços são mantidas secretas mesmo na presença de intrusões. Os resultados de avaliação mostram que a latência e débito do serviço proposto são similares aos de uma concretização de Kerberos bem conhecida.

Abstract. The Kerberos v5 standard specifies how the clients and services of a distributed system may mutually authenticate through the use of a centralized authentication service. If this service fails, by crash or in an arbitrary way (e.g., software bug, hardware problem, intrusion), the clients and services that depend on it are not able to authenticate between themselves. This paper presents an authentication and authorization service that complies with RFC 4120, and that uses Byzantine-fault-tolerant state machine replication and secure components to make the service more resilient. These secure components guarantee that clients' and services' secret keys are kept private even in the presence of intrusions. The evaluation results show that the proposed service has similar latency and throughput values to the ones of a well known Kerberos implementation.

1 Introdução

Hoje em dia, os sistemas informáticos da maioria das organizações utilizam algum tipo de serviço de autenticação e autorização de forma a impôr politicas de controlo de acesso a diferentes tipos de dados e/ou serviços. A norma Kerberos v5 propõe uma especificação para um serviço de autenticação (Kerberos) com duas características interessantes: faz uso de autenticação mediada, sendo portanto escalável do ponto de vista do número de chaves que cada entidade do sistema tem de armazenar, e apenas utiliza criptografia simétrica que é, como se sabe, mais rápida do que criptografia assimétrica.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 649–660

No entanto, o serviço de autenticação Kerberos é centralizado e concretizações comuns deste serviço tendem a residir apenas em um processo e em uma máquina. Basta que esse processo ou máquina falhe para que o serviço de autenticação se torne indisponível (falha por paragem) ou errático (falha arbitrária). Também basta a um intruso comprometer esse mesmo processo ou máquina para obter todas as chaves de todos os clientes e serviços do sistema. A falha do serviço de autenticação pode forçar todo o sistema informático que depende dele a parar, ou até mesmo permitir a utilização do sistema sem autenticação, dependendo de como o sistema foi concretizado e do tipo de falha do serviço de autenticação. Para além disto, a norma Kerberos v5 só especifica um serviço de autenticação, não fazendo qualquer referência à forma como o controlo de acesso/autorização é efectuado. Na prática, os serviços que concretizam esta norma (e.g., Apache DS, Microsoft Active Directory) combinam o serviço de autenticação Kerberos com um serviço de nomes e directorias (e.g., LDAP) onde é armazenada a política de controlo de acesso.

Este artigo apresenta o serviço de autenticação e autorização TYPHON[1], um serviço que obedece à especificação Kerberos v5, mas que é construído de forma a tolerar intrusões, fazendo uso da técnica de replicação da máquina de estados - para tolerar faltas arbitrárias - e de um componente seguro - para assegurar a confidencialidade das chaves no caso de acontecerem intrusões.

2 A Especificação Kerberos v5

O Kerberos v5 [8] é uma norma especificada no RFC 4120 [9]. Esta por sua vez é concretizada por várias aplicações, como o ApacheDS [1] e o Microsoft Active Directory[2].

Esta norma permite comunicações seguras e identificadas entre duas entidades por cima de uma rede insegura - como é o caso da Internet - e possibilita que duas entidades (tipicamente cliente e serviço) que à partida não têm nada que prove que podem confiar uma na outra, consigam autenticar-se mutuamente.

Para assegurar a confidencialidade dos dados e a autenticidade de clientes e serviços, é usada criptografia simétrica. Todos as entidades partilham uma chave secreta com o Kerberos e este faz de mediador entre as duas entidades que se pretendem autenticar.

A autenticação de uma entidade perante outra é conseguida por intermédio de estruturas de dados produzidas pelo Kerberos, denominadas de *tickets*. Os *tickets* provam que o Kerberos autenticou a entidade que pretende comunicar com outrem.

O Kerberos está dividido em dois componentes lógicos: o *Authentication Service* (AS) e o *Ticket Granting Service* (TGS). O primeiro permite a autenticação de uma entidade perante o Kerberos. O segundo destina-se a mediar a autenticação entre duas entidades após ambas estarem autenticadas perante o Kerberos (AS).

[1] Na mitologia grega, TYPHON é o pai de Cerberos (ou Kerberos).

[2] De notar que o ApacheDS e o Microsoft Active Directory são servidores LDAP que disponibilizam igualmente um serviço de autenticação Kerberos v5.

2.1 Interacções

A figura 1 ilustra[3] as interacções entre um cliente C, os componentes AS e TGS do Kerberos, e um serviço S.

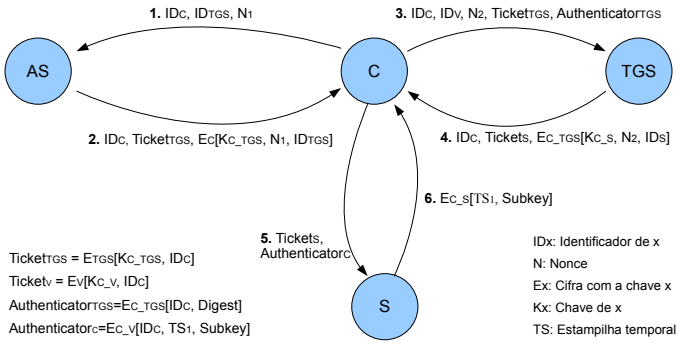


Figura 1. Ilustração simplificada das interacções das entidades presentes no Kerberos v5.

O objectivo das interacções ilustradas na figura 1 é autenticar o cliente C perante o serviço S. No passo 1 o cliente C requisita ao AS um *ticket granting ticket* (TGT). Esse TGT só poderá ser decifrado pelo Kerberos. No passo 2, o cliente obtém o TGT (Ticket_{TGS}) mais a chave de sessão contida nele (K_{C_TGS}) e produz um *authenticator* cifrado com essa mesma chave ($\text{Authenticator}_{TGS}$).

No passo 3, o cliente C envia Ticket_{TGS} e $\text{Authenticator}_{TGS}$ ao TGS, de forma a provar a sua identidade. Também indica o serviço com o qual se pretende autenticar, neste caso S. No passo 4, se o TGS conseguir decifrar $\text{Authenticator}_{TGS}$ com a chave contida em Ticket_{TGS} e encontrar S na sua base de dados, gera um *ticket* para S (Tickets) contendo uma chave de sessão a ser usada entre C e S (K_{C_S}). Por fim, no passo 5 e 6 o cliente C autentica-se perante o serviço S de forma semelhante à que se autenticou ao TGS.

3 Desafios na Concretização de um Kerberos Tolerante a Intrusões

A norma Kerberos foi criada com um pressuposto muito forte: o processo que concretiza a norma e a máquina que o executa nunca são comprometidos. O único perigo está na rede, que não assegura por si só a confidencialidade das mensagens. Mas tal assumpção não pode ser dada como garantida na prática. Isso dá origem a um ponto fraco nesta arquitectura: se o processo que executa o AS/TGS falhar por paragem, torna-se impossível realizar autenticação a partir do momento em que a falha ocorre. Os clientes que já tenham adquirido *tickets* para serviços ainda conseguem dialogar com os serviços para os quais esses *tickets* foram gerados[4], mas não é possível fazer novas autenticações.

[3] Esta ilustração é muito simplificada. Para facilitar a compreensão do protocolo, omitimos detalhes como o uso de *realms*, *flags* e o tempo de validade e renovação dos *tickets*.

[4] Mas não eternamente, porque os *tickets* têm um prazo de validade.

Também é possível que ocorram falhas arbitrárias, como por exemplo falhas de hardware, bugs e intrusões. No caso das intrusões, as chaves secretas podem ser obtidas, e quem se apoderar delas pode vir a personificar clientes e serviços.

O desafio neste trabalho é tornar mais robustas as quatro propriedades de segurança de um serviço Kerberos v5: autenticidade, integridade, confidencialidade e disponibilidade. A propriedade de autenticidade já é oferecida à partida pelo serviço Kerberos v5, uma vez que o seu objectivo é precisamente garantir a autenticidade dos clientes e serviços que o usam. Para garantir as três propriedades de segurança restantes, usamos a técnica de replicação de máquina de estados combinada com um componente seguro local em cada réplica.

4 Técnica de replicação de máquina de estados

Consideremos uma aplicação cliente/servidor, onde o servidor possui um estado, que evolui consoante os comandos enviados pelo cliente. A técnica da replicação da máquina de estados tem por objectivo replicar esse estado em vários servidores [11]. Esses servidores são considerados réplicas.

Os clientes enviam os seus comandos para essas réplicas, e o estado das mesmas deverá evoluir exactamente da mesma forma. Para isso cada réplica tem que começar a sua execução no mesmo estado, executar sobre ele apenas operações deterministas (e.g., não é possível gerar valores aleatórios, nem usar estampilhas temporais) e todos os comandos devem ser entregues na mesma ordem às réplicas - para este último requisito, é necessário uma primitiva de difusão atómica [6].

No entanto, é preciso notar que esta técnica de replicação, apesar de reforçar as propriedades de integridade e disponibilidade, não assegura confidencialidade e autenticidade. Pelo contrário, torna estas propriedades mais fracas. Isto porque ao tornarmos o sistema replicado, aumentamos a probabilidade de algum servidor ser corrompido. Como o estado está replicado em todos os servidores e, no caso do Kerberos o estado corresponde às chaves, basta uma intrusão para que estas sejam obtidas.

5 Protecção das Chaves

Para evitarmos o problema que acabámos de descrever, é necessário garantir que as chaves armazenadas por cada réplica não fiquem acessíveis mesmo quando a réplica for comprometida. No nosso sistema, a protecção das chaves é efectuada através da utilização de um componente seguro [5]. Um componente seguro consiste numa parte do sistema que se assume ser imune a intrusões, mesmo que o resto do sistema seja comprometido. Deverá ter uma especificação curta, de maneira a que a sua concretização seja simples, para ser exequível verificar as suas propriedades funcionais e não funcionais.

O componente seguro permite-nos obter a propriedade de confidencialidade, uma vez que armazena todos os dados confidenciais, i.e., as chaves. Também se delegam a este as operações que processam esses dados. A ideia é guardar os dados confidenciais no componente e executar operações sobre eles sem que estes sejam expostos ao

resto do sistema. Na secção 6.3 será explicado como é que este componente assegura a autenticidade.

6 O Serviço TYPHON

Nesta secção descrevemos em detalhe o serviço TYPHON, um serviço de autenticação e autorização tolerante a intrusões. Este serviço foi construído usando a primitiva de difusão atómica oferecida pela biblioteca BFT-SMaRt [2] e um conjunto de funcionalidades disponibilizadas por um componente seguro designado κ. De notar que conseguimos conceber este serviço sem subverter a especificação do Kerberos v5 tal como é apresentada no RFC 4120.

6.1 Modelo de Sistema

Assumimos um modelo e arquitectura de sistema híbridos em que o sistema é composto por duas partes, com propriedades e pressupostos distintos [12]. Estas duas partes designam-se tipicamente por *payload* e *wormhole*. Assume-se que a parte *payload* é composta por um conjunto de $n \geq 3f + 1$ réplicas e que um máximo de f réplicas podem falhar de forma arbitrária. Na parte *wormhole* executa um componente seguro com as funcionalidades que descreveremos mais à frente. Assume-se que esta parte do sistema é segura por construção, i. e., não pode ser comprometido por intrusos - desde que não exista acesso físico à máquina em que esse componente executa. assumesse que este componente pode falhar por paragem, mas se é só se a réplica associada ao componente estiver comprometida.

A biblioteca BFT-SMaRt executa na parte *payload*, assim como os componentes AS e TGS do serviço TYPHON. Assumimos que esta parte executa sob um modelo de sistema eventualmente síncrono semelhante ao definido em [4] uma vez que a primitiva de difusão atómica oferecida pela biblioteca BFT-SMaRt necessita deste pressuposto. Assumimos também sincronia local, i.e., o tempo máximo de processamento local e a taxa de desvio de cada relógio local são limitados e conhecidos. Estes pressupostos são requeridos pelo serviço de estampilhas temporais do BFT-SMaRt. Este serviço é usado pelo TYPHON tal como será explicado mais à frente.

6.2 Descrição Geral

O AS e o TGS apesar de serem componentes lógicos distintos são executados pelo mesmo processo. Por sua vez este processo é replicado em várias máquinas, e estas fazem uso da técnica da replicação da máquina de estados concretizada pelo BFT-SMaRt.

Se considerarmos que as chaves armazenadas pelo AS e TGS nunca são alteradas, o Kerberos v5 torna-se num sistema *stateless* (i.e., o estado nunca é modificado), e portanto poderia ser concretizado sem o uso da técnica de replicação de máquina de estados. Mas o TYPHON não tem esta característica e requer a manutenção de estado consistente nas suas réplicas por pelo menos três razões: (1) o serviço de autorização pode ser *stateful* na medida em que pode ter em conta autorizações prévias de um cliente

para decidir se este deve ou não aceder um novo serviço; (2) por razões de contabilização (*accountability*) e não-repudiação, o TYPHON armazena um histórico de todos os pedidos recebidos e respectivos resultados, e tal histórico precisa de ser consistente nas diversas réplicas; e (3) como foi mencionado na secção 4, a geração de estampilhas temporais não é uma operação determinista, e existe necessidade de estampilhas consistentes entre as réplicas. Para isso é necessário executar um acordo sobre o valor de cada estampilha que se queira gerar. A biblioteca BFT-SMaRt já disponibiliza essa funcionalidade.

Cada réplica tem acesso a um componente seguro. Esse componente guarda dentro de si as chaves dos clientes e dos serviços e oferece todas as operações essenciais para realizar operações com elas. Cada componente seguro possui ainda uma chave secreta que é usada para cifrar dados, gerar chaves de sessão e criar MACs para uma estrutura de dados especial que são os *ticket-approvals* (TAs).

TAs são estruturas de dados geradas pelo componente seguro de cada réplica para garantir que a geração, renovação e validação de um *ticket* para um serviço é permitida segundo a politica de autorização definida no TYPHON[5]. Quando os componentes TGS das várias réplicas do TYPHON recebem um pedido de *ticket* de serviço, cada réplica pede ao seu componente seguro a geração de um TA e envia esse TA para todas as outras réplicas. Posto isto, cada réplica espera a recepão de $2f + 1$ TAs, incluindo o TA dela própria. Se de entre $2f + 1$ TAs existirem pelo menos $f + 1$ válidos, significa que pelo menos uma réplica correcta está a autorizar a geração do *ticket* para esse serviço, logo o componente seguro de cada réplica pode também gerá-lo. Só são necessários $f + 1$ TAs válidos porque este passo de validação não é mais do que uma operação de leitura, logo, de entre esses $f + 1$ TAs válidos pelo menos um foi necessariamente gerado por uma réplica correcta, o que significa que a politica de autorização permite a geração do *ticket* de serviço.

Para assegurar a sua autenticidade, os TAs incluem um MAC gerado com a chave secreta dos componentes seguros - que é a mesma para todos. Também incluem dentro de si um resumo criptográfico dos parâmetros a serem passados ao componente seguro para a geração do *ticket* de serviço, de forma a assegurar que um conjunto de $f + 1$ TAs válidos só pode ser usado para gerar o *ticket* de serviço a que estão associados. Dentro de cada TA também está incluído uma estampilha temporal que evita ataques por repetição.

As chaves de sessão geradas por κ são aleatórias na medida em que são usadas as estampilhas temporais dos TAs e o valor da chave secreta de κ para criar essas chaves. As estampilhas garantem que o valor gerado é sempre diferente do anterior - isto é importante para não se gerar duas chaves com o mesmo valor. A chave de κ garante que esse valor não se consegue prever, pois a chave está dentro de κ e é secreta, logo, tornasse inexequível prever que valores vão ser gerados.

Todas as funcionalidades que não necessitem de realizar operações com/ou sobre as chaves são delegadas ao AS e TGS, como por exemplo definição de flags, verificação de realms, cálculo da validade dos *tickets* a serem emitidos, e outras operações semelhantes.

[5] Tal politica pode consistir simplesmente numa matriz de acesso que associa a cada cliente quais os serviços que este tem autorização para aceder.

Finalmente, existem três alternativas para integrar autorização num serviço kerberos, tal como especificado em [7]: no TGS, num serviço à parte, ou nos serviços destino. A terceira alternativa oferece mais flexibilidade, permitindo que cada serviço tenha a sua forma de controlo de acesso. Esta alternativa é de facto mais atraente num sistema super-distribuído com serviços de vários tipos e com diferentes níveis de criticidade. Mas no TYPHON estamos a assumir a primeira alternativa, que fará sentido num sistema distribuído homogéneo, e é esse tipo de sistema que estamos a assumir. Além disso, o controlo centralizado facilita a gestão da política de segurança do sistema.

6.3 Componente Seguro κ

Na concepção de um componente seguro é necessário ter em consideração o seguinte: (1) a quantidade de código no componente (é mais fácil verificar a sua correcção); (2) a interface do componente (é mais fácil usar e verificar que essa interface não pode ser usada para fazer acções erradas); e (3) a quantidade de vezes que o componente é acedido (porque como reside numa máquina separada, é tipicamente lento acedê-lo).

Tendo estas regras em consideração, concretizámos o componente seguro κ, que o AS e TGS devem utilizar sempre que precisarem de fazer operações que envolvam as chaves dos clientes e/ou serviços. Existe uma instância de κ por cada réplica do sistema. κ foi concebido para oferecer as operações mínimas necessárias a executar sobre *tickets* que envolvam as chaves. κ também tem uma chave secreta - guardada dentro de κ e só conhecida por ele - que usa para cifrar chaves de sessão quando são devolvidas para o resto do sistema. Desta maneira não é necessário guardá-las em κ para preservar a sua confidencialidade. Essa chave secreta também é usada para gerar os MACs dos TAs. Como tal chave está armazenada dentro de κ e este não pode ser comprometido, essa chave mantém-se segura. As operações de κ estão expostas na tabela 1.

Método	Argumentos	Retorno	Sumário
makeTicketApproval	*timestamp, client_name, server_name, hash_request*	*ticket_approval*	Gera *ticket-approvals*.
encryptParts	*enc_sessionKey, ticket_data, reply_data, ticket_approvals*	*ticket, reply_enc_part*	Gera o *ticket* e a parte cifrada da resposta do kerberos.
decryptParts	*ticket, authenticator, server_name*	*ticket_data, auth_data, enc_sessionKey*	Decifra o *ticket* e o *authenticator* enviados pelo cliente.
decryptPreAuth	*PrincipalName, pa_data*	*byte_array*	Decifra a pré-autenticaçao do cliente.

Tabela 1. Operações disponibilizadas por κ em cada réplica.

A operação *makeTicketApproval* gera um *ticket-approval* (TA) contendo a estampilha temporal, nome de cliente que requisita o *ticket*, nome de serviço ao qual o *ticket* se destina, e o resumo criptográfico passados à função. Estas estruturas de dados foram

explicadas na secção 6.2. Nesta função é usada a chave secreta de κ para produzir o MAC de cada TA emitido.

A operação *encryptParts* é usado na geração de *tickets* e na renovação/validação dos mesmos. Também precisa de receber *2f + 1* TAs gerados pelos componentes κ das outras réplicas, e pelo menos *f + 1* desses TAs precisam de ser válidos para se poder gerar o *ticket*.

A operação *decryptParts* é essencial para decifrar *tickets* e *authenticators*. A chave de sessão contida no *ticket* passado como argumento é devolvida cifrada com a chave secreta de κ, de forma a evitar que seja guardada dentro dele.

Finalmente, a operação *decryptPreAuth* serve para verificar a pré-autenticação do cliente.

6.4 Interacções

O algoritmo do TYPHON cumpre a especificação do Kerberos v5 e define um conjunto de interacções adicionais com um componente seguro. Ou seja, as interacções entre clientes e serviços, assim como as interacções entre clientes e AS/TGS continuam iguais ao Kerberos v5. No entanto, AS e TGS têm de contactar κ sempre que efectuem operações relacionadas com as chaves (confidenciais) de clientes e serviços. A figura 2 ilustra estas interacções.

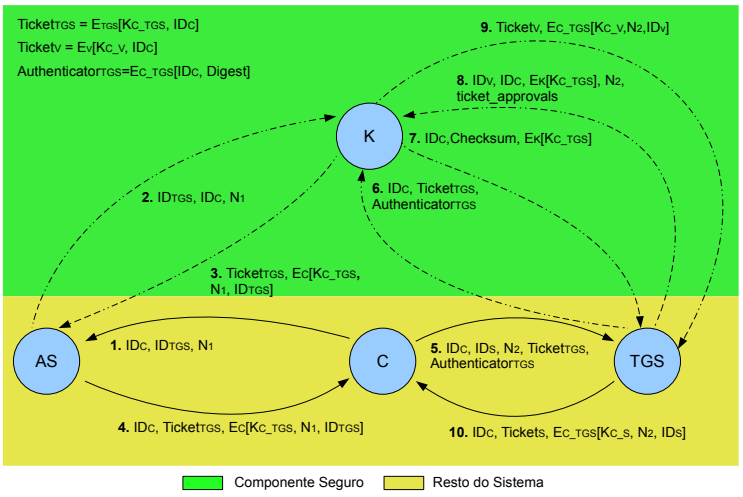


Figura 2. Ilustração simplificada das interacções das entidades presentes no TYPHON. A interacção com o serviço foi omitida desta figura porque é igual à da figura 1. A pré-autenticação também foi omitida por se tratar de um passo opcional segundo o RFC 4120.

No passo 1, C envia um pedido ao AS para obter um TGT. No passo 2, o AS reenvia os parâmetros desse mesmo pedido para κ, para que este crie um TGT e o tuplo cifrado a enviar a C.

No passo 3, após criar a chave de sessão (K_{C_TGS}), κ devolve ao AS o TGT ($Ticket_{TGS}$) e o tuplo cifrado ($E_C[K_{C\_TGS}, N_1, ID_{TGS}]$). No passo 4, o AS devolve a C o TGT e o tuplo.

No passo 5, C envia um pedido ao TGS para obter um *ticket* para S. No passo 6, o TGS re-envia para κ o ID de C (ID_C), o TGT ($Ticket_{TGS}$) e o *authenticator* ($Authenticator_{TGS}$). No passo 7, κ decifra estas duas estruturas e devolve o seu conteúdo ao TGS, com o cuidado de cifrar a chave de sessão contida no TGT com a sua própria chave secreta ($E_K[K_{C\_TGS}]$) - pois se o TGS for comprometido por um agente malicioso, este teria acesso a essa chave.

Após o TGS terminar as operações sobre os dados do TGT/*authenticator* e calcular novos dados de acordo com o RFC 4120, irá invocar κ para este criar um *ticket* para S. Esta interacção está representada nos passos 8 e é semelhante àquela que cria o TGT, com a diferença de que é especificado o identificador de S em vez do identificador do TGS, e também é fornecido $E_K[K_{C\_TGS}]$ de forma a que κ consiga criar $E_{C\_TGS}[K_{C\_V}, N_2, ID_V]$. Também é neste passo que todas as réplicas do sistema geram *ticket-approvals* e os enviam para as outras, para provar a κ que C tem autorização para interagir com S e por isso o *ticket* pode ser gerado.

No passo 9, κ devolve ao TGS o *ticket* para S (ID_S) e o tuplo cifrado que contém a chave de sessão entre C e S ($E_{C\_TGS}[K_{C\_S}, N_2, ID_S]$). No passo 10, estes são finalmente entregues pelo TGS a C. A partir daqui, a interacção entre C e S é igual à do Kerberos normal.

As interacções apresentadas aqui visam preservar a confidencialidade das chaves. Como κ é um componente seguro, as chaves nunca serão reveladas ao exterior, mesmo que no limite todas as réplicas sejam comprometidas. Isto porque κ não tem nenhuma operação que devolva as chaves. Os únicos dados confidenciais que o κ devolve são chaves de sessão, que vêm cifradas com a chave secreta deste. Se o tempo de utilidade dessas chaves (de sessão) for mais curto que o tempo que é necessário para quebrar a sua cifra através de um ataque de força bruta, também não é possível que intrusos consigam obter essas chaves em tempo útil. Também não se consegue gerar *tickets* para serviços arbitrariamente se o sistema for comprometido, devido ao uso de *ticket-approvals*, tal como explicado nas secções 6.2 e 6.3. Desta forma, conseguimos reforçar as propriedades de confidencialidade e autenticidade.

Note-se que os TAs não seriam necessários se estivéssemos a assumir que o TYPHON seria apenas um serviço de autenticação, deixando de lado a autorização. Os TAs servem para provar que existe pelo menos uma réplica correcta a fabricar esses *tickets*, o que significa que o cliente que requisitou esse *ticket* tem permissão para contactar o serviço que deseja. Mesmo que um intruso invadisse uma réplica e usasse κ para produzir *tickets* falsos em nome de outro cliente, este não conseguiria obter a chave de sessão que deve ser usada para fabricar o *authenticator* que deverá ser apresentado ao serviço para o qual se quer autenticar. Como não teria um *authenticator* para apresentar, não conseguiria fazer-se passar pelo cliente legítimo. Também não conseguiria usar ataques de repetição, pois a própria especificação do Kerberos está feita para se defender disso.

7 Avaliação

Nesta secção comparamos a latência e o débito de TYPHON e ApacheDS. O ApacheDS é um serviço de nomes e directorias, mantido pela Apache, que concretiza especificações como LDAP e Kerberos v5.

O TYPHON e o ApacheDS estão ambos escritos na linguagem Java. Como já foi referido, o TYPHON usa a bibliteca BFT-SMaRt, que também está escrita em Java. O TYPHON tem 2943 linhas de código (não contando as 7557 linhas de código da biblioteca BFT-SMaRt) e apenas 8,7% são executadas pelo componente seguro.

O nosso ambiente de testes consistiu em um conjunto de 5 máquinas interligadas por um switch gigabit. Para avaliar o TYPHON, replicá-mo-lo em quatro máquinas ($n = 4$) de forma a tolerar uma falta ($f = 1$) em alguma dessas 4 máquinas. A restante máquina executa todos os processos cliente, que estão constantemente a pedir um TGT seguido de um *ticket* de serviço. O ApacheDS foi avaliado de forma semelhante, com apenas numa máquina a executar o serviço e tendo um cliente noutra máquina a enviar-lhe pedidos da mesma forma que no caso do TYPHON.

Efectuámos dois tipos de experiências. Na primeira avaliámos a latência fim-a-fim da geração de um TGT seguido da geração de um *ticket* de serviço. Na segunda experiência medimos a quantidade de TGTs e *tickets* de serviço que se conseguem gerar por segundo. Estas experiências foram efectuadas em ambos os sistemas e os resultados são descritos na próxima secção.

7.1 Latência

Os resultados dos testes de latência são exibidos na tabela 2. Para cada sistema, foram efectuados 20.000 pedidos de TGTs e *tickets* de serviço, tendo-se calculado a média da latência dos últimos 10.000 pedidos. Os primeiros 10.000 pedidos foram utilizados para forçar a máquina virtual Java a fazer uso do compilador JIT (*Just-In-Time*) para transformar em código máquina as instruções do ApacheDS e TYPHON que tratam dos pedidos de TGTs e *tickets* de serviço.

TYPHON		ApacheDS	
TGT	*Ticket* de serviço	TGT	*Ticket* de serviço
4.6	5.7	26.3	3.0

Tabela 2. Resultados dos testes de latência para ambos os sistemas (em milisegundos).

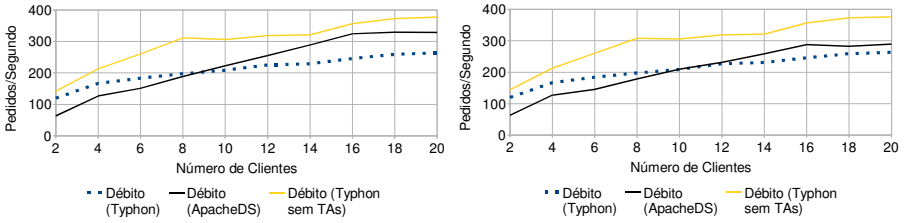
Os resultados da latência dos *tickets* de serviço mostram que o TYPHON é mais lento a realizar essa operação do que o ApacheDS. Por outro lado, podemos observar que o ApacheDS é bastante mais lento do que o TYPHON na geração de TGTs. Este resultado é inesperado, mas sabemos que enquanto o TYPHON guarda todos os seus dados em memória depois de ser inicializado, o ApacheDS vai buscá-los ao disco. Isto explica que a latência do ApacheDS seja maior que a do TYPHON[6].

[6] No entanto, após contactar a equipa de desenvolvimento do ApacheDS, ficámos a saber que este devia guardar essa informação numa *cache* em memória. Eles próprios fizeram os seus testes de latência e confirmaram os nossos resultados, mas ainda precisam de investigar o que está a acontecer que explique este fenómeno.

A diferença entre os valores de latência do TYPHON para TGTs e *tickets* de serviço é explicada pelo facto dos TGTs não requererem a utilização de TAs, enquanto que os *tickets* de serviço requerem. A latência média do envio e recepção de TAs corresponde portanto a 1.1 ms, a diferença entre a latência dos *tickets* de serviço (5.7 ms) e a latência dos TGTs (4.6 ms). No entanto é necessário ter em consideração que numa situação real o TYPHON teria que gerar mais *tickets* de serviços do que TGTs, e o ideal seria que fossem os *tickets* de serviço a ter uma latência menor.

7.2 Débito

Para computar o débito, calculámos o número de pedidos por segundo processados por ambos os sistemas em intervalos de 1000 pedidos. Como a escalabilidade pretendida é em termos de clientes, aumentámos progressivamente o número de clientes de forma a observar o ponto de saturação de cada sistema. A figura 3 reporta os resultados.



(a) Débito da geração de TGTs.

(b) Débito da geração de *tickets* de serviço.

Figura 3. Resultados dos testes de débito para os serviços ApacheDS e TYPHON (com e sem *ticket-approvals*).

Os gráficos mostram um comportamento semelhante tanto no caso de TGTs como de *tickets* de serviço, e sugerem que quando se trata de poucos clientes a enviar em simultâneo, o TYPHON mostra melhor comportamento que o ApacheDS. Mas a partir de 10 clientes, o TYPHON estagna, e o ApacheDS passa a apresentar melhores resultados. No entanto, a partir de 20 clientes também o ApacheDS estagna, apesar de se manter mais eficiente que o TYPHON.

Finalmente, fizemos um último teste retirando o uso de TAs da concretização do TYPHON, tornando-o somente num serviço de autenticação. Observámos que a remoção deste passo faz com que o sistema se torne mais eficiente a processar pedidos, chegando ao ponto de mostrar melhores desempenhos que o ApacheDS.

8 Trabalhos Relacionados

No passado foram propostos alguns serviços de segurança tolerantes a intrusões (e.g., autoridade de certificação COCA [13], serviço de gestão de chaves Ω [10], firewall aplicacional CIS [3]). No entanto, tanto quanto é do nosso conhecimento, este trabalho é o primeiro a apresentar um serviço de autenticação e autorização tolerante a intrusões que cumpre a norma Kerberos v5.

9 Conclusão

O Kerberos v5 é uma norma que especifica como clientes e serviços se devem autenticar mutuamente por intermédio de uma entidade centralizada que guarda as chaves de todos os participantes. O problema reside na possibilidade dessa entidade centralizada falhar, quer seja por paragem, arbitrariamente ou até por intrusão. Se isso acontecer não é possível fazer mais autenticações de clientes ou serviços.

Neste artigo apresentámos o serviço TYPHON, um serviço de autenticação e autorização que segue a especificação do Kerberos v5 ao mesmo tempo que introduz na sua concretização mecanismos para tolerar intrusões. Por um lado, usa a técnica de replicação da máquina de estados para oferecer tolerância a faltas arbitrárias. Por outro lado, faz uso de componentes seguros para guardar as chaves dos clientes e dos serviços de forma a assegurar que estes não são expostos na eventualidade de intrusões.

Finalmente, os resultados da avaliação do serviço TYPHON mostram que o seu desempenho é similar ao do ApacheDS, um serviço de autenticação e autorização não replicado que concretiza a norma Kerberos v5.

Agradecimentos. Este trabalho é suportado pela FCT através de seu programa multianual (LaSIGE) e através dos projectos PTDC/EIA-EIA/100581/2008 (REGENESYS) e CMU-Portugal (CMU-PT/0002/2007).

Referências

1. ApacheDS - An embeddable directory server entirely written in Java. `http://directory.apache.org/`.
2. BFT-SMART - High-performance Byzantine fault-tolerant State Machine Replication. `http://code.google.com/p/bft-smart/`.
3. A. Bessani, P. Sousa, M. Correia, N. F. Neves, and P. Verissimo. The CRUTIAL way of critical infrastructure protection. *IEEE Security & Privacy*, 6(6):44–51, Nov-Dec 2008.
4. M. Castro and B. Liskov. Practical Byzantine fault-tolerance and proactive recovery. *ACM Transactions Computer Systems*, 20(4):398–461, Nov. 2002.
5. M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Proc. of the 4th European Dependable Computing Conference*, Oct. 2002.
6. V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical report, Cornell University, May 1994.
7. S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Section E.2.1 kerberos authentication and authorization system, Apr. 13 1989.
8. B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, Sept. 1994.
9. C. Neuman, S. Hartman, and K. Raeburn. The kerberos network authentication service (v5). Internet Engineering Task Force RFC 4120, July 2005.
10. M. K. Reiter, M. K. Franklin, J. B. Lacy, and R. N. Wright. The Ω key management service. In *Proc. of the 3rd ACM Conf. on Computer and Communications Security*, 1996.
11. F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22, Dec. 1990.
12. P. Verissimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1):66–81, 2006.
13. L. Zhou, F. Schneider, and R. Van Rennesse. COCA: A secure distributed online certification authority. *ACM Transactions Computer Systems*, 20(4):329–368, Nov. 2002.

Web Application Risk Awareness with High Interaction Honeypots *

Sergio Nunes[1] Miguel Correia[2]

[1] Novabase [2] Universidade de Lisboa

Abstract. Many companies are deploying their business on the Internet using web applications. Risk awareness allows to mitigate the security risk of these applications. This paper presents an experiment with a collection of high interaction web honeypots in order to analyze the attackers' behavior. Different security frameworks commonly used by companies are analyzed to evaluate the benefits of the honeypots security concepts in responding to each framework's requirements and consequently mitigating the risk.

Resumo. Muitas empresas estao a lançar o seu negócio na Internet usando aplicações web. O *risk awareness* permite mitigar o risco associado a essas aplicações. Este trabalho apresenta uma experiência com um conjunto de honeypots web de alta interação, de modo a analisar o comportamento dos atacantes. Diferentes *security frameworks* utilizadas por empresas são analisadas para avaliar os benefícios do uso de honeypots web no contexto da mitigação de risco.

1 Introduction

Nowadays, most of the traffic circulating in the Internet is web traffic, traveling over the HTTP and HTTPS protocols. As multiple applications are moving to the web with the Web 2.0 phenomenon, this type of traffic tends to increase. The Web provides unified access to dynamic content with a simple browser, being able to encapsulate and integrate multiple technologies. There are multiple web ramifications divided among multiple browsers, webservers, web languages and databases that must all function flawlessly together, despite the involved complexity. The development of such web applications in its own is a complex task. Developers suffer pressure regarding time to market minimization and this leads to time sparing in software testing procedures. Without adequate security testing, web applications are deployed with multiple vulnerabilities. The data accessed through web applications is becoming more and more critical, containing private information that enables financial transactions in multiple online businesses. This vicious cycle is growing and organizations are unable to foment the necessary risk awareness to be able to analyze these new web threats.

* This work was partially supported by the FCT through the CMU-Portugal partnership and the Large-Scale Informatic Systems Laboratory (LaSIGE).

This new massification of web technologies poses multiple questions regarding information security: What is the role of security with this significant change? Is there an improvement in the confidentiality, integrity and availability of information with this new situation? Are there any new security threats that put information at risk?

The objectives of this paper are to address these questions by implementing a *high-interaction honeypot environment* composed of several common web applications used in the Internet that have reported vulnerabilities. By exposing these vulnerable web applications in a monitored honeypot architecture, the attacks can be captured and investigated, along with the tools and actions of the attacker after the intrusion. The proactive honeypot deceptive techniques record as close as possible the attackers' behavior to minimize his/her advantage, instead of relying in the common prevention, detection and reaction security approach, in the usual situation of waiting to be attacked. The careful analysis of the detailed gathered attack data and the know-how gained by managing honeypots, provides an insight about the modus operandi and motives of the attacker, classifying him according to a pre-established profile.

Having the attacker profile defined, the threat model can be specified in order to develop the necessary risk awareness and risk mitigation controls. Risk mitigation is accomplished in organizations by employing a variety of information security, compliance and risk *frameworks* that address multiple domains across the wide information technology environment. The paper considers three frameworks: ISO/IEC 27001 , Cobit and PCI-DSS. These frameworks present a major focus in security guidelines by providing specific control requirements and objectives to control risk in organizations integrating people, processes and technology as a whole. These frameworks present most of the time general guidelines that do not descend to specific security technologies, so it is important to evaluate how common security technology concepts adapt to these frameworks. Honeypots can bring added value to such frameworks by satisfying multiple enumerated control requirements.

In a nutshell, the paper tackles its objectives in a sequence of three steps:

1. Recollection of attack data using a high-interaction honeypot environment with several common web applications;
2. Web application attackers profiling based on the data obtained in step 1;
3. Analysis of the honeypots' benefits to the security guidelines provided in common risk assessment frameworks, based on the results of steps 1 and 2.

2 Context and Related Work

The honeypots' main function is to be probed and attacked [15,1,3,13]. The value of this security mechanism relies on monitoring the real steps and tools of a real attack and learning where the unknown vulnerabilities lie and how to protect the critical information assets. These monitoring and decoy capabilities aid the security professional in developing the required know-how of the modus operandi

of the attacker and infer the security situational awareness of his network to plan for the adequate safeguards and effective incident responses [16].

Web honeypots are means for gathering web attack information and develop situational risk awareness [6,14]. The Google Hack Honeypot (GHH) [10] reveals a new use for honeypots as it simulates vulnerable web applications that are commonly searched by attackers over search engines. The attacking search procedure uses careful placed search queries that are able to find vulnerable applications by matching specific strings in the previous indexed information. Mueter et al. developed a toolkit for converting automatically PHP applications into high-interaction honeypots [11]. They tested the Honeypot-Creator against a wide variety of applications and analyze the results using their high interaction analysis tool (Hihat).

What is the risk to business operations of an attack happening? Most of the time, this question remains unanswered in organizations that have services and do business over the Internet. It is crucial to mitigate the security risk using common frameworks of risk management and compliance. The regulatory compliance that organizations must meet should be dealt with due care by the upper business management, so it is necessary to have an effective way of controlling and securing information technologies. Nowadays there are multiple compliance and risk frameworks so the question remains which to use and where to direct its efforts to achieve adequate risk mitigation.

The ISO/IEC 27001 is an international standard that provides a model for establishing an Information Security Management System (ISMS) as a strategic organization decision [8]. The objective of an organization by being certified in this standard is the compliance that it has put effective information security processes in place, instead of applying non repeatable ad-hoc procedures. The certification issued by an independent third party serves as evidence that the security controls exist and function according to the standard requirements. This evidence can serve as advantage against competitors, can respond to the compliance requests of some costumers and assures business security following best practices which generate a trust relationship.

The Information Systems Audit and Control Association published the Control Objectives for Information and Related Technology (Cobit) to help information technology governance professionals to align technology, business requirements and risk management [7]. Cobit is positioned at the higher business management level dealing with a broad range of IT activities and focuses on how to achieve effective management, governance and control.

The Payment Card Industry Data Security Standard (PCI-DSS) was developed to assure cardholder data security and unify consistent data security measures globally [12]. It was created by American Express, Discover Financial Services, JCB, MasterCard Worldwide and Visa International to establish requirements for the security of the payment card industry affecting everyone that stores card payment data, including common online commercial transactions.

3 Web Application Honeypots

3.1 Honeypot environment

This section deals with the planning, implementation, configuration and analysis of the high interaction honeypot environment. The main requirement for this environment was the ability to gather detailed attack and malicious action information that provided a real situational risk awareness regarding web attacks. The environment had to be similar to a real production deployment. The option chosen was to deploy a virtual high interaction honeynet, because it does not limit the attacker's actions. The testbed was composed by real operating systems, webservers, databases and web applications constrained by virtualization.

The honeypots network, also known as honeynet, had to be managed remotely under secure conditions due to the high monitorization that this sort of high interaction honeypots needs. The solution relied on the use of a management station with SSH access over the Internet [4].

Minimize the management burden was another requirement that is tackled with the deployment of VMware Server that allows transparently copying and moving of honeypot virtual machines. The possibility of emulating ISO images as a virtual cd-rom also accelerates the installation process. VMware Server also provides the possibility of deploying checkpoints to be able to return to previous states if the honeypots are compromised or intermediate state forensic analysis is needed.

There is the risk of the attacker targeting other systems after honeypot compromise, so this situation must be controlled and safeguarded as a requisite. The response was the use of Honeywall, a layer 2 bridge with filtering, attack detection and connection limiting capabilities between the honeynet and the Internet and the possibilities of monitorization of the virtual honeypot in the host operating system employing the principle of security layering by employing multiple approaches [2].

The hardware used was composed by 12 Dell and Fujitsu Siemens Pentium 4 and Core 2 Duo PCs with 512MB to 2GB of RAM. One PC was used as the Honeywall bridge, another was the management console and the remaining ones were the VMware Server hosts used for the honeypots. The management and honeypot networks used two dedicated HP Procurve 2600 series switches physically separated. The software used for the honeypot host systems was a minimal installation of Ubuntu 8.04 which is the most recent version supported by VMware Server 2.0.1.

The honeypot host systems had two network interfaces (NICs): one configured static IP address for management and the other configured with access to VMware without IP address. The management is performed over SSH and via the VMware management console over the management NIC. The xtail command line utility was installed and configured for watching the VMware virtual disk files. Monitoring of the honeypots was done using Sebek, a kernel module designed by the Honeynet project for that purpose [15,5].

The honeypots implemented used different operating systems, different webservers, different databases and different web applications developed in different languages, as can be seen in Table 1. The operating system choice division was based on compatibility with Sebek and representativeness in the Internet hosts commonly used as webservers. The name of the honeypots represent the operating system installed with "webserver" for the Linux machines, "xp" for Windows XP machines and "win2003" for Windows 2003 machines.

Honeypot Name	Operating System	Webserver	Database	Application
Webserver1	Ubuntu 7.10	Apache 2.2.4	Mysql	PHPbb
Webserver2	Ubuntu 7.10	Apache 2.2.4	-	Wordpress
XP1	Windows XP	Apache	Mysql	EasyPHP
Win2003	Windows 2003	IIS 6.0	SQLServer	Snitz Forum
Webserver3	Ubuntu 7.10	Apache 2.2.4	Mysql	PHPNuke
Webserver4	Ubuntu 7.10	Apache 2.2.4	Mysql	PHPmyadmin
Webserver5	Ubuntu 7.10	Apache 2.2.4	Mysql	PHP-fusion
XP2	Windows XP	IIS 5.1	-	ASP-CMS
XP3	Windows XP	Tomcat	-	JSP Examples

Table 1. Honeypots specification

3.2 Experimental results

This section presents an overall statistical analysis of the results gathered from the honeypot environment from June to September of 2009 with the analysis of the attack information across different detailed graphs.

Figure 1 shows that during this time frame our environment suffered a total of 8858 attacks. It can also be observed that the first honeypot named "webserver1" (see Table 1) suffered more attacks that the other honeypots. This can be explained by its position in the IP address range as the first host serving HTTP requests as a webserver. Detecting the availability of a webserver, the attacker starts by targeting automatically this host with all his arsenal of web exploits without checking the installed web application and gives up without probing sequentially the next IP address.

The large majority of the attacks detected were not specific to the applications installed, but randomly or sequentially scanned across the honeypot IP address range for multiple specific vulnerabilities. The number of targeted attacks is 498 representing only 6% of the total of targeted and untargeted attacks.

The diversity of operating systems and webservers present in our honeypot environment does not influence the attack number results as there is no significant distinction on attack rate by operating system or when comparing webserver technologies as it can be observed in Figure 1.

Attacks to web applications (Figure 2) reveal that PHP is the most attacked web language with PHPMyAdmin as the most attacked application, while the

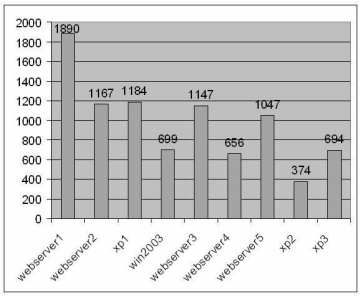


Fig. 1. Number of attacks by honeypot (8858 total)

other installed applications present no significant number of attacks with the exception of the tomcat manager. There is a significant amount of blind attacks to commonly used Internet web applications that were not installed in the environment like Horde, Roundcube or Zencart. These web applications are widely deployed over the Internet so attackers prefer to conduct random or sequential exploitation in order to compromise the highest number of machines possible with little target search and information gathering procedures.

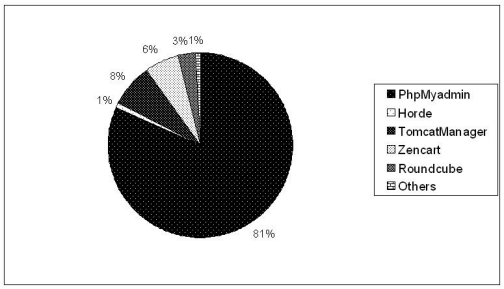


Fig. 2. Percentage of attacks by application

As it can be seen in Figure 3, there is a large amount of URL bruteforcing attacks, trying to find hidden applications with known vulnerabilities by enumerating default locations and version numbers. Direct command execution is also tried across multiple known vulnerable applications, because of the simplicity in compromising vulnerable hosts in this manner. Code Injection was accomplished against a known vulnerability in PHPMyadmin and remote file include was tried in requests to non existent vulnerable web applications in our environment. Authentication bruteforce attacks were performed against the tomcat manager application.

Figure 4 shows the worldwide origin attack distribution that probed our environment based on source addresses using GeoIPlite country mapping database by Maxmind [9]. There were 272 different attack sources detected with an average of 32 attacks by country. The United States was the main source of attack of the environment followed by China as the new rising star in hacking attempts with their huge evolution in technological resources. The addition of

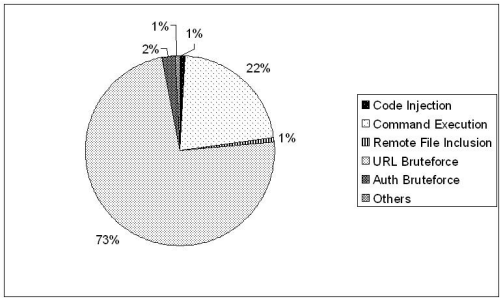


Fig. 3. Percentage of attacks by type

both these sources represents more than half of the attacks verified in the honeypot testbed. The diversity of attacking countries captured by the environment shows that there are attackers almost everywhere that try to intrude systems over the Internet bypassing any geographical borders, language barriers and cultural issues. There were only 9 attacks detected from Portuguese sources, which consisted only of web server fingerprinting attempts.

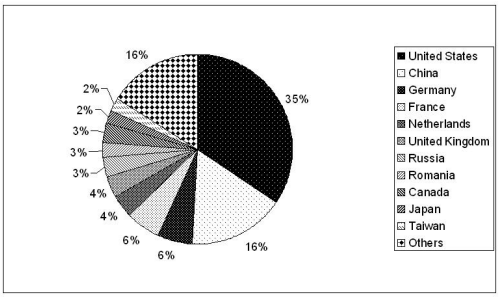


Fig. 4. Top attacking countries

Some of these attack sources can be innocent hosts that were previously intruded and are used as remote headquarters for conducting further attacks. The wide search for open proxies verified in the honeypot testbed also shows these resources are being used to masquerade the real source of attacks.

Comparing these results with the statistics of recent web attacks, we can conclude that that there was no attempt to exploit multiple cross site scripting and SQL injection vulnerabilities present in our environment, as these vulnerabilities require more knowledge to adapt to the attacker's final objective. The major threat of information leakage was not verified in our environment as it does not present real sensitive information. It can be verified that our environment suffered a high number attacks that show a rise of web threats, but as the number of targeted attacks is low it is impossible to see a wide variety of attack and vulnerability types. The high number of untargeted attacks suffered by our environment dictates that there is a maximization of quick intrusion efforts by probing the entire Internet address space for a recent disclosed vulnerability.

3.3 Attacker profiling

Based on the data and evidence gathered in our honeypot environment, this section deals with profiling the attackers of our environment, describing the characteristics and modus operandi that allow recognizing their behavior.

Most of the attacks that we faced were driven by *script kiddies* testing the latest disclosed exploit globally throughout the Internet, without even first fingerprinting the web server to see if it runs the vulnerable application. They were apparently driven by pure curiosity as most of them replayed the published exploit without any code changes and repeated its execution multiple times when, in some cases, there was no possibility of success. Most of them jumped the necessary information gathering and scanning phase to try directly to get access to the supposed vulnerable system. The intrusion can be easily identified as most of these individuals do not have sufficient skills to erase effectively their tracks or remain undetected inside the host. Their attacks are untargeted as they sweep multiple host ranges using the disclosed exploit sequentially with no focus on the system as a whole or its data value, but only as a single IP address inside the range chosen. Others performed enumeration tasks in the scanning phase looking for specifically unprotected administration components using published scripts and tools. When those components were found with authentication requirements, they conducted default and common user and password enumeration. This behavior reveals a more practical knowledge with proficiency in the use of malicious attacking tools, being able to analyze the results provided by them. As the results show failure in exploitation or take to much time to complete, they jump to the next system without analyzing further ways of intrusion.

A minority of attacks has evidence of *bot owners* as they have a modus operandi similar to script kiddies, but their main motivation is to install a bot to control the target remotely. They also start directly in the gaining access phase by searching for a specific vulnerability along a predefined range of IP addresses to maximize the intrusions and consequently the number of bots installed. After identifying a successful intrusion they upload, install and hide the bot automatically using an automated deployment script. The remote bot management is performed using an alternative protocol such as IRC, having possibilities of upgrading the bot software and of performing manual commands on the compromised host. Another difference in this modus operandi when comparing with script kiddies is that they are worried in hiding the bot and remain undetected, by for example disabling the anti-virus or installing a rootkit, in order to maintain the access to their zombies active and continue increasing the botnet power and size. This botnet power and size are the main factors that influence the profit when selling the botnet in the black market, if financial gain is the attacker's major motivation.

Our honeypot infrastructure is installed in a university IP range and has no real challenge regarding data value. The honeypot applications installed tried to simulate confidential data value such as students' forums, blogs and administration panels with predefined known vulnerabilities. Any knowledge attacker will first gather information about the target and conclude that it is situated in a

university and unless he has specific reasons to attack that host, he will continue his challenge elsewhere. The only event for which we can conclude that the attacker gathered information about the IP range ownership was the attempt to proxy requests to a scientific subscription article site. The attacker researched that multiple universities have access to scientific subscription article sites and some of those sites authenticate the subscription with the universities source IP address providing access to paid articles. The motive of this attack can be classified as profit to save money by not buying the individual articles directly onsite or selling this privileged information to other individuals looking to access the scientific subscription articles for less money than the online subscription.

4 Risk Awareness

There are multiple frameworks commonly used by organizations that help us to organize an information security system measuring the risk involving IT assets. This section analyzes how the honeypots can contribute to the risk awareness concerning threat and vulnerability identification by looking at multiple frameworks in a methodological critical approach. Using the knowledge gathered from the honeypot testbed experience and the profiling of the attacker's mindset, an evaluation is performed to research how the honeypot concepts adapt to each framework's objectives and controls, bringing added value to the organization's risk mitigation requirements.

The ISO/IEC 27001 standard mandates to monitor and review the ISMS to identify attempted and successful security breaches and incidents. The honeypots could bring to this requirement increased added value when compared to traditional intrusion detection systems, because of the detailed information gathered about an attack, which enables gaining real know-how and situational awareness of the risk that the asset faces. The usual intrusion detection systems deployed in organizations commonly match attack signatures with attacking procedures full of false positives and deviate the time of security personnel from protecting the critical assets.

In ISO/IEC 27002, the supporting standard for ISO/IEC 27001, there are some controls that can be adapted to the added value of honeypots. The control for protection against malicious code (27001 Annex A.10.4.1) can be complemented with a honeypot by performing evaluation of malicious code using client honeypots and by having a honeypot infrastructure capable of monitoring malicious code spreading mechanisms. The use of multiple different malware analysis is suggested in the standard as a vector to improve the effectiveness of malicious code protection.

The ISO/IEC 27002 standard suggests that is necessary to reduce risks from exploitation of technical vulnerabilities (27001 Annex A.12.6). The control defines that timely information about technical vulnerabilities of information systems being used should be obtained, the organization's exposure to such vulnerabilities evaluated and appropriate measures taken to address the associated risk. This is the main focus of the honeypot technology and by adequate use of

honeypots it is possible to accomplish this goal of establishing an effective management process for technical vulnerabilities that responds to the requirements.

The ISO/IEC 27002 standard details the need to ensure a consistent and effective approach to the management of information security incidents (27001 Annex A.13.2.2). It suggests defining the responsibilities and procedures to deal with the incidents collecting forensic evidence for internal problem analysis. The forensic evidence can also be used to pursue a legal action preserving the chain of custody that assures the admissibility in court. This collection of evidence can be gathered using honeypots or honeypot data gathering mechanisms. It can be seen that the chain of custody has multiple requirements to be admitted in court, so training how to collect and preserve the evidence should be an exercise first performed on decoy systems such as honeypots, to prepare for a real incident on production systems.

The ISO/IEC 27002 standard states that there should be a learning experience from information security incidents allowing the incidents to be monitored and quantified. The information gained from the evaluation of information security incidents should be used to identify recurring or high impact incidents. This learning can be developed with the risk and threat awareness delivered with the continuous use and analysis of honeypots. Honeypots were created precisely as a mechanism for learning about the modus operandi of attackers.

In the ISO/IEC 27002 standard there is a section concerning the correct processing in applications (27001 Annex A.12.2) detailing components such as input and output data validation that are the cause of multiple web attacks like those analyzed in this paper. Although honeypots are no direct defense against those attacks, they provide the necessary learning and research capabilities necessary for secure programming and correct evaluation of the risk that results with the lack of validation in applications. The attacked decoy web applications can measure the threat level and serve as case studies for future applications developed.

The protection of organizational records is also a subject detailed in the ISO/IEC 27002 standard regarding its loss, destruction or manipulation (27001 Annex A.12.5.4). Organization information disclosure attacks happen frequently in an enterprise and they are difficult prevent or even to detect. The concept of honeytokens can help in the detection of disclosure of critical data by placing careful bogus monitored records in such datastores and track those records while they travel through the network serving as a warning that the data is being disclosed.

A similar analysis has been done to COBIT and PCI-DSS, but it is not possible to show it for space reasons. Table 2 summarizes the results of the analysis done for the three frameworks.

It can be observed in the table that the honeypots can bring benefits to multiple requirements in each framework. More generically, the major benefits of using honeypot concepts when dealing with risk frameworks are:

- The creation of a risk awareness culture being able to correctly identify the threats to IT and evaluate the impact to business of attacks;

Honeypot Concept	**ISO/IEC 27001**
Risk Awareness	4.2 Establishing and managing the ISMS
Secure Coding	A.12.2 Correct processing in applications
Malicious Code Detection	A.10.4.1 Controls against malicious code
Information Disclosure Detection	A.12.5.4 Information leakage
Vulnerability Management	A.12.6 Technical vulnerability management
Incident Response	A.13.2.2 Learning from information security incidents
Honeypot Concept	**COBIT**
Risk Awareness	PO9 Assess and manage IT risks
Secure Coding	AI2 Acquire and maintain application software
Malicious Code Detection	DS5.9 Malware prevention, detection and correction
Information Disclosure Detection	DS11.6 Security requirements for data management
Vulnerability Management	DS5.5 Security testing, surveillance and monitoring
Incident Response	DS5.6 Security incident definition
Honeypot Concept	**PCI-DSS**
Risk Awareness	12.1.2 Identify threats and vulnerabilities, conduct risk assessment
Secure Coding	6.5 Develop all web applications with secure coding guidelines
Malicious Code Detection	5.1.1 Detect, remove and protect against malware
Information Disclosure Detection	3.1 Keep cardholder data storage to a minimum
Vulnerability Management	6.2 Identify newly discovered security vulnerabilities
Incident Response	12.9 Implement an incident response plan

Table 2. Honeypot benefits to three frameworks studied

- The promotion of secure coding by learning from the application attacks suffered, evaluating the coding vulnerabilities that were explored and developing the safeguards necessary to correct them;
- The detection of malicious code due to monitorization of propagation attempts and unusual activity, along with the testing of suspicious webpages and binaries in a test decoy environment;
- The detection of disclosure of information with the monitorization of decoy bogus items (honeytokens);
- The creation of an accurate and timely vulnerability management framework being able to identify, analyze and patch with a minimum time delay recently disclosed exploits and malicious tools used by attackers;
- The creation of an incident management and response system capable of identifying, classifying and addressing security problems;

5 Conclusion

In this paper an evaluation of web attack threats is presented focusing in the importance of developing risk awareness to mitigate them. To gather this attack information, a high-interaction web honeypot environment was installed, configured and monitored during approximately 4 months. This research confirmed our previous belief that honeypots are useful for companies but underestimated

by them, probably mainly because of a lack of knowledge regarding this technology, its uses and benefits. The fear of challenging the attacker and being unable to control the consequences of the intrusion is also a deterrence factor in the use of honeypots by companies. These issues are never balanced with the possibility of developing the necessary risk awareness within the company using these decoy systems to be able to defend the critical assets when a real attack emergency happens. We believe this is a critical factor enhanced by the use of honeypots: the possibility of being familiar with the modus operandi of the attacker and being prepared to respond to a real situation. Readiness only becomes effective with adequate training and this training is done using a test honeypot environment.

References

1. Anagnostakis, K.G., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E., Keromytis, A.D.: Detecting targeted attacks using shadow honeypots. In: Proceedings of the 14th USENIX Security Symposium (2005)
2. Chamales, G.: The Honeywall CD-ROM. Security & Privacy, IEEE 2(2), 77–79 (March-April 2004)
3. Dagon, D., Qin, X., Gu, G., Lee, W., Grizzard, J., Levine, J., Owen, H.: Honeystat: Local worm detection using honeypots. In: Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID). pp. 39–58 (2004)
4. Hatch, B.: SSH port forwarding. SecurityFocus http://www.securityfocus.com/infocus/1816 (January 2005)
5. Honeynet-Project: Know your enemy: Sebek (November 2003)
6. Honeynet-Project: Know your enemy: Web application threats (April 2008)
7. ISACA: Cobit framework 4.1. http://www.isaca.org (2007)
8. ISO/IEC 27001: Information technology - security techniques - information security management systems - requirements
9. Maxmind: Geoiplite. http://www.maxmind.com (2009)
10. McGeehan, R.: Ghh http://ghh.sourceforge.net/
11. Mueter, M., Freiling, F., Holz, T., Matthews, J.: A generic toolkit for converting web applications into high-interaction honeypots. University of Mannheim (2008)
12. PCI-DSS: Payment card industry data security standard version 1.2. http://www.pcisecuritystandards.org (October 2008)
13. Portokalidis, G., Slowinska, A., Bos, H.: Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. In: Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys). pp. 15–27 (2006)
14. Riden, J., Oudot, L.: Building a PHP honeypot. InfoSecWriters http://www.infosecwriters.com (April 2006)
15. Spitzner, L.: Honeypots: Tracking Hackers. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
16. Yegneswaran, V., Barford, P., Paxson, V.: Using honeynets for internet situational awareness. In: Proceedings of the 4th Workshop on Hot Topics in Networks (HotNets) (2005)

Sistemas Embebidos e de Tempo-Real

Exploiting AIR Composability towards Spacecraft Onboard Software Update

Joaquim Rosa, João Craveiro, and José Rufino

* Universidade de Lisboa, Faculdade de Ciências, LaSIGE

Abstract. The AIR architecture, developed to meet the interests of the aerospace industry, defines a partitioned environment for the development of aerospace applications, adopting the temporal and spatial partitioning (TSP) approach, and addressing real-time and safety issues. The AIR Technology includes the support for mode-based schedules, allowing to alternate between scheduling modes during a mission, according to different mission's operation plans. Furthermore, it can be necessary, useful or even primordial having the possibility to host new applications in the unmanned spacecraft onboard computer platform in execution time. In this paper we define the foundations of a methodology for onboard software update, taking advantage of the composability properties of the AIR architecture, in order to add new features to the mission plan.

Resumo. A arquitectura AIR, desenvolvida para responder aos interesses da indústria aeroespacial, define um ambiente compartimentado para o desenvolvimento de aplicações aeroespaciais que adoptem a abordagem de compartimentação temporal e espacial, discutindo questões de tempo-real e de segurança no funcionamento. A Tecnologia AIR inclui o suporte para alternar entre vários modos de escalonamento durante uma missão, de acordo com diferentes planos de funcionamento. Além disso, pode ser necessário, útil ou mesmo primordial ter a possibilidade de alojar novas aplicações ou funcionalidades no computador de bordo do veículo espacial não-tripulado em tempo de execução. Neste artigo definimos os fundamentos de uma metodologia para actualização de software durante o funcionamento do sistema, aproveitando as propriedades de componibilidade da arquitectura AIR, para adicionar novas funcionalidades ao plano da missão.

1 Introduction

Future space missions aiming long-term durations call for a new generation of spacecrafts. This has driven the interest from the space agencies and industry partners in the definition and design of fundamental building blocks for onboard computer platforms, where the strict demands for reliability, timeliness, safety and security are combined with an overall requirement to reduce the size, weight and power consumption (SWaP) of the computational infrastructure.

* This work was partially developed within the scope of the European Space Agency Innovation Triangle Initiative program, through ESTEC Contract 21217/07/NL/CB, Project AIR-II (ARINC 653 in Space RTOS — Industrial Initiative, http://air.di.fc.ul.pt). This work was partially supported by Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology), through the Multiannual Funding and CMU-Portugal Programs and the Individual Doctoral Grant SFRH/BD/60193/2009.

The definition of partitioned architectures implementing the logical containment of applications in criticality domains, named partitions, allows to host different applications in the same computational infrastructure and enables the fulfilment of those requirements [14]. The notion of temporal and spatial partitioning (TSP) ensures that the activities in one partition do not affect the timing of activities in other partitions and prevents the applications to access the addressing space of each other.

The AIR (ARINC 653 In Space Real-Time Operating System) Technology emerges as a partitioned architecture for aerospace applications [13] applying the TSP concepts. The AIR architecture allows the execution of both real-time and generic operating systems in independent partitions, ensures independence from the processing infrastructure and enables independent verification and validation of software components.

In a partitioned architecture, the several functions of an unmanned spacecraft, such as Attitude and Orbit Control Subsystem (AOCS), Telemetry, Tracking and Command (TTC) subsystem, share the same computational resources, being hosted in different partitions. Partitions are scheduled according to fixed cyclic scheduling tables. The AIR architecture allows the possibility to dynamically alternate between different scheduling tables. This is useful for the adaptation of partition scheduling to different mission operating modes and for the accommodation of component failures [13].

During the course of a mission, situations may appear on which it may be useful or even necessary to introduce new functions or to modify existing ones to deal with unexpected events. For example, in the presence of a failure of a specific component, it may be necessary to change the mission plan by reconfiguring the applications' scheduling. An example where such features had an important role was the incident with NASA's rover Spirit [4]. In May 2009 the rover was stuck on Mars soft sand terrain and after some months of trying to release it without success, the NASA's team decided to change the mission plan and instead of doing surface exploration, the rover started working as a stationary research platform, performing functions that would not be possible to a mobile platform, such as detecting oscillations in the planet's rotation which would indicate a liquid core.

The modular design of the AIR architecture and the separation of applications in the temporal and spatial domains enables composability properties which are exploited in the build and integration process. This means that the several components can be developed, verified and validated independently. To a software provider, this procedure does not depend on knowledge of the other partitions and, at most, is aided by guidelines to accomplish timeliness requirements. To the system integrator, it is assigned the responsibility of ensuring the accomplishment of system-wide temporal requirements. This paper addresses how to take advantage of the composability properties of the AIR architecture to establish the basis of an onboard software update methodology.

The remainder of this paper is organized as follows. In Section 2 we describe the AIR Technology including the schedulability and composability properties of the architecture, and the build and integration process. In Section 3, we describe the requirements, the components and the integration process of the onboard software update, along with the methodology defined. In Section 4, we expose future research directions and some related work. Finally, Section 5 concludes the paper.

2 AIR Technology Design

The AIR Technology design was original prompted by the interest of the European Space Agency (ESA) in the adoption of TSP concepts to the space industry. The AIR Technology is currently evolving towards an industrial product definition by improving and completing its architecture definition and engineering process [12,13].

2.1 System Architecture

The AIR architecture, illustrated in Fig. 1, allows applications to be executed in logical containers called partitions. At the application software layer (Fig. 1) applications consist in general of one or more processes, which make use of the services provided by an *Application Executive* (APEX) *interface*, as defined in ARINC 653 specification [1]. In addition, a system partition may invoke also specific functions provided by the core software layer, thus being allowed to bypass the standard APEX interface (Fig. 1).

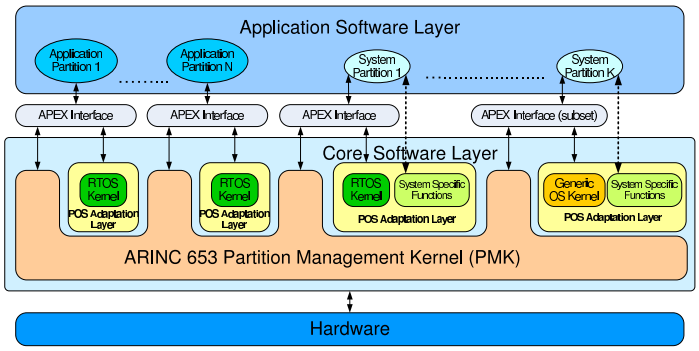


Fig. 1. AIR Architecture and Integration of Partition Operating Systems

The core software layer provides a (real-time or generic) operating system kernel per partition herein referred to as *Partition Operating System* (POS). The *AIR POS Adaptation Layer* (PAL) [6] wraps each POS, hiding its particularities from the AIR architecture components.

The AIR architecture implements the advanced notion of portable APEX, meaning portability between the different POSs is built on the availability of PAL related functions and on the APEX core layer, which may exploit the POSIX application programming interface available on most (real-time) operating systems. The APEX provides the required partition and process management services, time management services, intra-partition and inter-partition communication services and health monitoring services.

The partition management, inter-partition communication and health monitoring services rely additionally on the *AIR Partition Management Kernel* (PMK) service interface. The AIR PMK bears the most responsibility in ensuring robust TSP. The temporal partitioning is achieved by scheduling the partitions according to a given scheduling

table, repeated cyclically over a *major time frame* (MTF). The spatial partitioning is ensured by a high-level abstraction layer which provides a mapping between AIR protection requirements and the hardware's addressing space protection mechanisms.

The AIR architecture also incorporates a *Health Monitor* (HM) component to handle hardware and software errors, containing them within their domains of occurrence.

2.2 Temporal and Spatial Partitioning

To ensure the safety and timeliness of mission-critical systems and minimize the drawbacks arising from the integration of multiple functions sharing the same hardware resources, the design of AIR Technology proposes the architectural principle of robust partitioning. With partitioning we achieve two important properties. The first concerns containing the occurrence of faults to the context where they appear, and thus not interfering with the system overall behaviour. The other property has to do with system *composability* enabling the independent verification and validation of software components that also facilitates the overall certification process, fundamental for space-borne vehicles.

The AIR architecture has been designed to fulfil the requirements for robust TSP. Temporal partitioning ensures that the activities processed in one partition do not affect the real-time requisites of the functions running in other partition. Space partitioning relies on having separate addressing spaces and thus not allowing an application to access the memory and input/output (I/O) spaces of a different partition.

2.3 Designing for Schedulability

The original ARINC 653 [1] notion of a single fixed partition scheduling table, defined offline, is limited in terms of timeliness control and fault tolerance. The design of the AIR architecture incorporates the advanced notion of *mode-based partition schedules*, allowing temporal requirements to vary according to the mission's phase or mode of operation [13,2].

An AIR-based system includes a set of partition schedules among which it can switch during its operation. A schedule switch can be ordered by a specific partition designed and allowed to do so, through the invocation of an APEX primitive. This can, in turn, result from either a command issued from ground control or from reacting to environmental conditions as obtained by the spacecraft's sensors. The order will not come into immediate effect, but rather applied at the end of the current MTF.

The AIR Partition Scheduler component is responsible for guaranteeing that the processing resources are, at every time, assigned to the correct partition and for making schedule switch effective at the end of the respective MTF. Its implementation is described in pseudocode in Algorithm 1. This is executed at every system clock tick, inside the respective interrupt service routine. The implementation of this algorithm is optimized to introduce little overhead to such routine.

The first verification to be made is whether the current instant is a partition preemption point (line 2). In case it is not, the execution of the partition scheduler is over; this is both the best case and the most frequent one. If it is a partition preemption point, we

Algorithm 1 AIR Partition Scheduler featuring mode-based schedules

```
1:  ticks ← ticks + 1                                  ▷ ticks is the global system clock tick counter
2:  if schedules_currentSchedule.table_tableIterator.tick =
      (ticks − lastScheduleSwitch) mod schedules_currentSchedule.mtf then
3:      if currentSchedule ≠ nextSchedule ∧
          (ticks − lastScheduleSwitch) mod schedules_currentSchedule.mtf = 0 then
4:          currentSchedule ← nextSchedule
5:          lastScheduleSwitch ← ticks
6:          tableIterator ← 0
7:      end if
8:      heirPartition ← schedules_currentSchedule.table_tableIterator.partition
9:      tableIterator ← (tableIterator + 1) mod
          schedules_currentSchedule.numberPartitionPreemptionPoints
10: end if
```

then verify (line 3) if there is a pending scheduling switch to be applied and if the current instant is also the end of the MTF. If these conditions apply, then a different partition scheduling table will be used henceforth (line 4). The partition which will hold the processing resources until the next preemption point, dubbed the heir partition, is obtained from the partition scheduling table in use (line 8) and the AIR Partition Scheduler will now be set to expect the next partition preemption point (line 9).

2.4 Designing for Composability

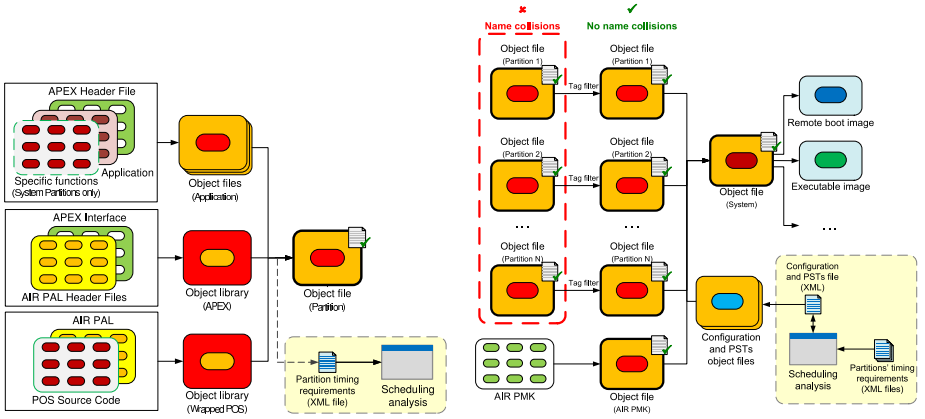
The design of the AIR architecture and the use of a TSP approach enables the *composability properties* of AIR-based systems, in both time and space domains. The use of a fixed cyclic partition scheduling scheme dictates that the timeliness guarantees of each partition are defined by the processing time assigned to each partition. In the spatial domain the composability properties ensure that the partition's memory and I/O resources are protected against unauthorized access from other partitions. The composability properties are thus inherent the AIR modular architecture.

The modularity of the AIR architecture design and of its build and integration process further enables the composability of AIR-based systems [5]. This means, on a first approach, that the several components that may compose such a system can be developed, verified and validated independently. This eases certification efforts, since only modified modules need to be reevaluated. It is also a fundamental basis for onboard software update as proposed in this paper.

From the point of view of one partition's provider, this further signifies that development and validation does not depend on knowledge of the other partitions (individually or as a whole). At most, the development of one partition should be aided by a set of guidelines for its applicability to the target TSP systems in general. The system integrator is responsible for guaranteeing a correct partition scheduling, so that partitions and the system as a whole meet their timing requisites [5].

2.5 Build and Integration Process

Because of the particularities of the architecture, the software build and integration process needs to differ from the canonical application build process, as provided by standard compilers and linkers. This process is pictured in Fig. 2 and it will now be described in detail.



(a) Software build by partition application developers

(b) System integration

Fig. 2. AIR build and integration process

Partition build process

The first stage concerns building each partition independently (Fig. 2a). In the typical scenario, the applications to be executed in the context of a partition, the APEX library, and the underlying POS libraries (wrapped by the AIR PAL) may be provided by different teams or providers. Therefore, the build process is tailored to expect these independent object files, and link them together to produce an object file with no unresolved symbols but including relocation information (to allow linking with the remaining partitions). Although the AIR PAL also invokes the AIR PMK (which symbols are as of yet undefined), these interactions are wrapped using data structures to reference the appropriate primitives, which the AIR PMK will register by executing code generated at system integration time with the assistance of a specific AIR tool.

The introduction of a scheduling analysis phase in the application developers' software production chain [5] takes advantage of the composability properties to provide independent schedulability analysis. Application developers can perform this analysis using the timing requirements (period, worst-case execution time, deadline, etc.) of their applications' processes. This information can be either estimated, or tentatively determined through static code analysis [11].

System integration

The system integration process (Fig. 2b) receives input (partition object files) from potentially different teams or providers. Since all partitions will include the common interface provided by the AIR PAL and AIR APEX libraries, the various partitions' object files will have symbol name collisions; partitions running the same POS or POSs providing the same standardized interfaces (e. g., POSIX) have additional name collisions. Therefore, linking these objects will require previous preprocessing. This preprocessing can be in the form of a *tag filter* utility which prefixes all symbols and calls in each partition's object files with unique prefixes (e. g., `P1`, `P2`, etc.). This process can be further optimized by automating the generation of partition prefixes, namely deriving them from the configuration file.

The partition objects can now be linked with the AIR PMK and the configuration object. This configuration object is derived by compiling C source code files, which in turn have been converted from XML (Extensible Markup Language) configuration files. The use of XML for the configuration file is motivated by the overall intention to comply, up to a certain degree, with the ARINC 653 specification [1]. Besides the parameters translated from these XML files (such as partition scheduling tables, addressing spaces, and inter-partition communication ports and channels), configuration objects include routines for the AIR PMK to register the adequate primitives in the AIR PAL structures. This linking step produces the system object file, from which in turn one can generate the most adequate deployment format for the target platform. In the system integration phase, scheduling analysis capabilities shall be introduced in relation with the generation of a system-wide configuration [5].

3 Onboard Software Update

We establish the foundations of a methodology to allow including new features on a spacecraft during a mission. The challenges we face are related to maintaining the real-time and safety guarantees defined for the original mission. Adding a new application to the system should be performed in a way that does not affect the overall behaviour of the system, including the timeliness of the already running applications.

3.1 Defining Requirements and Components

To support the upload of modified software components to the spacecraft's onboard computer platform, we assume the existence of a (secure) communication channel and a data communication protocol. The communication functions aboard the spacecraft are responsible for dealing with the reception of the data sent by the ground station and for performing online processing of the transferred data stream. Handling the update of onboard software components implies: the identification of the components being updated (partition software components, PSTs sets); the allocation of the required memory resources; the functional integration of each component in the operation of the onboard computer platform. The onboard software update handler shall be implemented as an activity (process/thread) in the domain of the (system) partition associated to the communication functions.

To the partition hosting the communication functions it is ensured a given time processing budget. However, we assume that software update activities are performed on a best-effort basis, thus with minimal impact on the timeliness of the communication functions. This ensures the safety of onboard software update since it will not interfere with other communication functions, namely with the detection and the identification of ground commands.

To support the introduction of onboard software update operations, the original APEX interface must be extended with the services presented in Table 1. However, only the APEX interface of the partition hosting the onboard software update functions needs to be extended.

Table 1. Extended APEX services for Onboard Software Update

Primitive	**Short description**
XAPEX_MALLOC	Allocate memory from the partition's free memory pool
XAPEX_MFREE	Deallocate a memory zone for the partition's free memory pool
XAPEX_MCLAIM	Claim memory from a specified partition for the partition's free memory pool
XAPEX_PUPDATE	Apply partition software components update
XAPEX_PSTUPDATE	Apply system partition scheduling table (PST) set update

3.2 Integration on Spacecraft Onboard Platform

We assume the component dedicated to onboard software update, the Update Handler, is defined as a process/thread integrated in the partition responsible for the communication functions, as illustrated in the simplified spacecraft architecture [8], pictured in Fig. 3. This partition also includes a command detection function. Commands issued from ground mission control will be passed to the TTC through a inter-partition communication channel. One example is a ground command to change a PST.

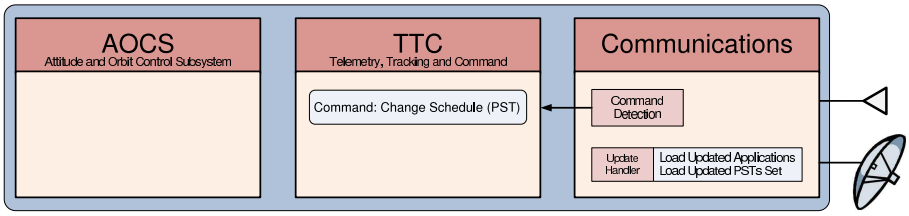


Fig. 3. Spacecraft onboard platform

3.3 Designing an Onboard Software Update Methodology

The design of a methodology for onboard software update in AIR-based systems has evolved from the build and integration process. This methodology is extended to cope with the modification of software components in order to upgrade the original mission.

This may include the modification of application software, partition or system wide configurations or simple the definition of a new set of partition scheduling tables (PSTs). The complete methodology consists in a four-step procedure as follows:

STEP 1: Offline Verification and Validation of Software Modifications

The modifications to the software components of a given mission may include the redesign of the applications associated with a given partition (e.g., payload functions) and the definition of a new set of PSTs. The linking of the modified partition with the objects of other partitions is made on the logical address space in order to guarantee that the mapping of unmodified partitions remains unchanged. This way, only the updated components need to be uploaded to the spacecraft onboard computer platform. This process is illustrated at the left side of Fig. 4 and may involve scheduling analysis of the partition. The update of the mission may simply involve the modification of a given PSTs set. In this case, the schedulability analysis and the generation of a new configuration and PSTs set is only performed at the system integration stage.

This corresponds to the AIR original verification and validation process of software components performed on the ground, before sending the applications to the spacecraft, and consists on applying the build and integration process to ensure that the safety and the TSP requirements would not be compromised with the introduction of new components on the system. Due to the composability properties of the AIR architecture, the build process may be done by the software development teams or providers independently. Each team or provider, along with the new application, delivers the partition timing requirements, that altogether will form the partition scheduling tables (PSTs), used by the AIR PMK Partition Scheduler on the target system.

The output produced in this step is the system object file, resulted from the integration of all the built objects potentially from various developers.

STEP 2: Extraction of Updated Components

After having the result of the build and integration process done on the previous step, there is the need to identify which components need to be uploaded to the spacecraft onboard computer platform. The final goal of this step is to identify those components, extract them from the complete system object file and create a new object composed only by the components to be uploaded to the spacecraft onboard computer. Also, it is necessary to build the object file according to a specific format, in order to the Update Handler be able to recognize the data received and perform its handling.

Like the previous one, this step is made on the ground. It requires a special-purpose toolset to perform the extraction and the formatting functions. The extraction and the formatting actions are represented by the shaded area at the right side of Fig. 4.

Finally, the updated object will be uploaded to the spacecraft using the communication facilities to exchange data between the ground stations and the space vehicles.

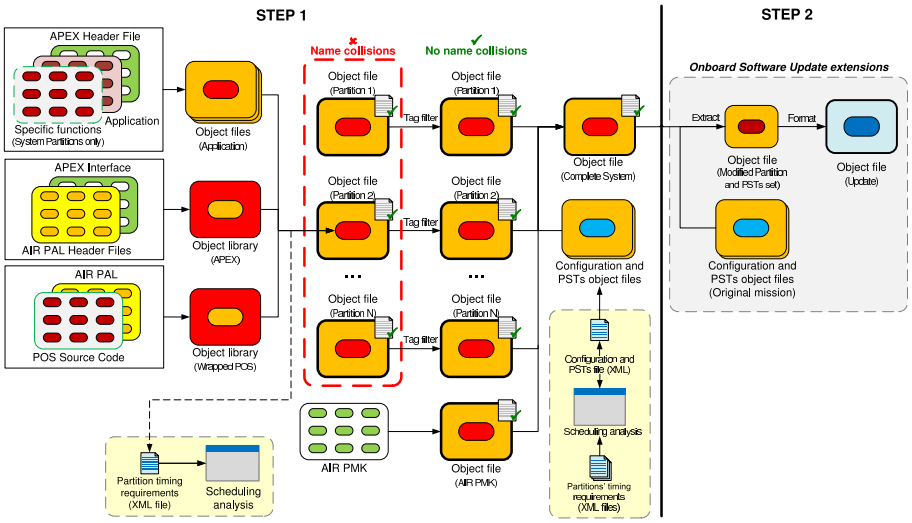


Fig. 4. Integration of an AIR-based system extended with the extraction and formatting of the updated components

STEP 3: Transfer of Updated Components

In the spacecraft, the application and PSTs uploaded in a single object file are received by the partition running the application responsible for the communication operations. Complementarily to the formatting done in the Step 2, when the modified components were formatted into an object file, the Update Handler look into the uploaded object file and separate the application of the PSTs.

We assume the existence of a component which will provide the required communication facilities between the spacecraft and the ground stations.

Upon reception of partition software components the Update Handler will invoke the XAPEX_MALLOC primitive to allocate the required memory. We assume that the available memory is large enough to contain the updated application. The Update Handler may also invoke the XAPEX_MCLAIM primitive to claim the memory used by the partition being updated, followed by the XAPEX_PUPDATE primitive which assigns the updated software components to the specified partition (Table 1). Finally, upon reception of a PSTs set, the Update Handler will invoke the XAPEX_PSTUPDATE primitive which will apply the PST set update.

STEP 4: Activation of Updated Components

To guarantee that applying the updated PSTs set does not compromise the safety of the whole mission, the XAPEX_PSTUPDATE (Table 1) will perform a blocking wait until the proper conditions are met, as described in Algorithm 2. The first condition for safe application of a new set of PSTs is that the currently selected schedule is identical in both the existing and the updated PSTs sets. The second condition is that a schedule

switch to a PST which has been modified in the updated set is not pending. The goal of these conditions is to ensure that the operation conditions that the system expects and/or the criteria by which the system or a ground operator has chosen the current or next schedule are not voided.

Algorithm 2 XAPEX_PSTUPDATE primitive

1: **while** $schedules_{currentSchedule} \not\equiv newSchedules_{currentSchedule} \vee$
$schedules_{nextSchedule} \not\equiv newSchedules_{nextSchedule}$ **do** ▷ Wait (block)
2: **end while**
3: SWAP($schedules$,$newSchedules$)

After the new PSTs have been activated, the uploaded partition application can now be scheduled, a situation which may occur upon receiving a schedule switch command from the ground mission control, as illustrated in Fig. 3.

4 Future Developments and Related Work

The importance of a strong verification and validation process in critical systems is addressed in [3] and the relevance of a safety-policy validation at binary level is highlighted in [10]. The problem of dependable online upgrade of real-time software was approached in [16].

The methodology established in this paper for onboard software update can be further extended to cope with the upgrade of critical software components that must be performed without interruption, such as those ensuring AOCS, TTC and communication functions. This implies a new set of challenges to be addressed specifically in the steps 3 (transfer of updated components) and 4 (activation of updated components). Although driven by the specific requirements of aerospace applications, these developments may benefit from the work performed on dynamic software update [7,9,15,17].

Solutions for dynamic software update on real-time systems requiring the identification of specific points in time for components' update is discussed in [17], while [15] makes no presumption about new application's periods and execution times.

The results achieved in [7] shown that the real-time and fault-tolerant requirements of avionics systems could be accomplish even during a dynamic reconfiguration of the system due to component failures. An approach for dynamic update of applications in C-like languages is provided in [9] and focuses on the update of the code and data at predetermined times, but does not specify real-time requirements.

5 Conclusion

In this paper we described the AIR Technology, towards build aerospace applications to temporal and spatial partitioning systems. Motivated by the need to add new applications in the system during a mission, due to changing its plans, we defined the onboard software update requirements and discussed how to take advantage of the composability

inherent to the build and integration process of the AIR-based systems. We establish a methodology for onboard software update, that exploits the composability properties of the AIR architecture, allowing independent verification and validation. The onboard software update methodology is based on the redefinition of the original space mission and it is supported on a specific toolset for the extraction of the updated software components, to be uploaded to the spacecraft onboard computer. The methodology can be further extended to support dynamic update of critical software components.

References

1. AEEC (Airlines Electronic Engineering Committee): Avionics application software standard interface, part 1 - required services. ARINC Specification 653P1-2 (Mar 2006)
2. AEEC (Airlines Electronic Engineering Committee): Avionics application software standard interface, part 2 - extended services. ARINC Specification 653P2-1 (Dec 2008)
3. Bahill, A.T., Henderson, S.J.: Requirements development, verification, and validation exhibited in famous failures. Systems Engineering 8(1), 1–14 (2005)
4. Brown, D., Webster, G.: Now a Stationary Research Platform, NASA's Mars Rover Spirit Starts a New Chapter in Red Planet Scientific Studies. http://www.nasa.gov/mission_pages/mer/news/mer20100126.html (Jan 2010)
5. Craveiro, J., Rufino, J.: Schedulability analysis in partitioned systems for aerospace avionics. In: Proc. 15th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA 2010). Bilbao, Spain (Sep 2010)
6. Craveiro, J., Rufino, J., Schoofs, T., Windsor, J.: Flexible operating system integration in partitioned aerospace systems. In: Actas do INForum - Simpósio de Informática 2009. Lisboa, Portugal (Sep 2009)
7. Ellis, S.M.: Dynamic software reconfiguration for fault-tolerant real-time avionic systems. Microprocessors and Microsystems 21, 29–39 (1997)
8. Fortescue, P.W., Stark, J.P.W., Swinerd, G. (eds.): Spacecraft Systems Engineering, 3rd Edition. Wiley (2003)
9. Hicks, M.: Dynamic software updating. ACM Transactions on Programming Languages and Systems 27(6), 1049–1096 (Nov 2005)
10. Necula, G.C., Lee, P.: Safe kernel extensions without run-time checking. In: Proc. USENIX 2nd Symposium on Operating Systems Design and Implementation. pp. 28–31 (1996)
11. Pushner, P., Koza, C.: Calculating the maximum execution time of real-time programs. Journal of Real-Time Systems 1, 160–176 (Sep 1989)
12. Rufino, J., Craveiro, J., Schoofs, T., Tatibana, C., Windsor, J.: AIR Technology: a step towards ARINC 653 in space. In: Proceedings of the DASIA 2009 "DAta Systems In Aerospace" Conference. EUROSPACE, Istanbul, Turkey (May 2009)
13. Rufino, J., Craveiro, J., Verissimo, P.: Architecting robustness and timeliness in a new generation of aerospace systems. In: Casimiro, A., de Lemos, R., Gacek, C. (eds.) Architecting Dependable Systems 7. LNCS, Springer, Berlin Heidelberg (2010), accepted for publication
14. Rushby, J.: Partitioning in avionics architectures: Requirements, mechanisms and assurance. Tech. Rep. NASA CR-1999-209347, SRI International, California, USA (Jun 1999)
15. Seifzadeh, H., Kazem, A., Kargahi, M., Movaghar, A.: A method for dynamic software updating in real-time systems. In: Proceedings of the 8th IEEE/ACIS International Conference on Computer and Information Science. Shanghai, PR China (Jun 2009)
16. Sha, L.: Dependable system upgrade. In: RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium. p. 440. IEEE Computer Society, Washington, DC, USA (1998)
17. Wahler, M., Ritcher, S., Oriol, M.: Dynamic software updates for real-time systems. In: Proceedings of the HotSWUp'09. Orlando, Florida, USA (Oct 2009)

Resilient Middleware for a Multi-Robot Team

Eric Vial, Mário Calha

FC/UL **
evial@lasige.di.fc.ul.pt
mjc@di.fc.ul.pt

Abstract. This paper addresses a resilient cooperative engine for robot teams within the context of the surveillance of physical areas. Due to the unreliability in the wireless communication between robots, a middleware must offer some resilience to the control application and guarantee that the robots never collide. We present an architecture for the robots to share a common view and to handle new events in a safe and resilient way. The system relies on two control sub-modules, the first one, the payload, could be complex and has access to information shared among robots, the second one, the wormhole is reliable but only uses local information. The system is evaluated by means of simulation tools and aims to be ported to hardware platforms composed by real mobile robots.

Resumo: Este documento aborda um motor cooperativo e resiliente para equipas de robôs no contexto da vigilância de áreas físicas. Devido à falta de fiabilidade na comunicação sem fios entre robôs, um middleware deve oferecer alguma resiliência à aplicação de controlo e garantir que os robôs nunca colidem. Apresentamos uma arquitectura que permite aos robôs partilharem uma vista comum e lidar com novos eventos de uma forma fiável e resiliente. O sistema apoia-se em dois sub-módulos de controlo, o primeiro, payload, pode ser complexo e acede à informação partilhada pelos robôs, o segundo, wormhole é confiável mas apenas utiliza informação local. O sistema é avaliado através de ferramentas de simulação e tem como objectivo ser implementado em plataformas de hardware compostas por robôs reais.

Keywords: group communication, middleware, mobile robots

1 Introduction

Mobile robot teams have the potential to reduce the need of human presence for complex or repetitive tasks. For most of them, the use of cooperation between robots can enhance the overall performance of the team. Achieving an efficient cooperation requires the use of complex algorithms implemented in each

** Faculdade de Ciências da Universidade de Lisboa. Bloco C6, Campo Grande, 1749-016 Lisboa, Portugal. Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by the FCT through the LASIGE Multiannual Funding Program.

robot. This internal complexity in addition to the interaction issue with the environment makes the robot control system more sensitive to failures. Nowadays, building resilient control systems for mobile robots is a real challenge.

In this paper, we will present a middleware architecture for robots in charge of monitoring a physical area. In particular, we will focus on three architecture features which improve the system resilience: First, a control layer which relies on an hybrid approach, both synchronous and asynchronous. This layer involves two sub-systems the *payload* and the *wormhole* and guarantees a timely execution of critical tasks. The second architecture feature is the tree-oriented event structure used in the *payload* and based on small computation modules, This structure offers more stability to the system by avoiding cycles of events. Finally, as last feature, we will describe a group of modules in charge of managing a common world view for all robots and in particular a synchronization algorithm used during group merging or splitting phases. The algorithm is designed to be tolerant to communication failures.

The project context is a cooperative surveillance application of a given area. The covered zone is a campus, a plant or any well-defined area, each robot has a prior map of this environment. The purpose is to detect an accident or an intrusion and to build a common strategy to handle properly the detected event (e.g. blocking the intruder). All robots run the same version of the middleware and are equipped with local sensors, positioning and wireless devices.

This paper is organized as follows: The next section addresses the related work. In section 3, we provide an architecture overview. Section 4 gives details on the *wormhole* and *payload* model, the event-based architecture, and the world view synchronization algorithm. This latter part includes a short analysis with pros and cons. Finally, section 5 describes pending and future applications of the work and concludes the paper.

2 Related work

In the last twenty years, there has been a considerable amount of work to study mobile robot localization. Researches have been carried out focussing on two problems: computing absolute location using a priori map [6] or building incrementally this map while exploring the environment [5]. Both approaches most often rely on complex and math-oriented algorithms based on Kalman filters and maximum likelihood estimation [7]. The present paper does not address this kind of problem and we assume that the robot is equipped with a location device based on GPS or RSS technology [1]. In the same way, the way a robot team performs the surveillance of a physical area could obey many rules in order to maximize the probability of locating an intruder [8]. We wilfully chose not to optimize this part, the robot just wanders around the world, making random decisions to turn left or right at every crossing.

In the control architecture, the payload relies on a flexible and modular tree-oriented architecture. The idea is to break down the control layer into a chain

[1] Global Positioning System and Received Signal Strength

of small modules. Each one is triggered with an incoming event and is able to generate outgoing events up to others modules. This concept is a simplified application of the subsumption theory developed by Rodney A. Brooks [9] at the beginning of the 80's. Unlike many implementation projects based on this theory, we do not allow information cycles between module layers, modules are top-down triggered which minimizes the risk of an out-of-control diffusion of events. Finally, we took up the idea of suppressing some input signal to inhibit a group of modules which is similar to the Brooks suppressor concept.

In the vehicular domain, designing safety-critical application is essential, the work in [1], [2] or [3] presents a hybrid (synchronous and asynchronous) control model used in real-time applications. This model is based on two sub-systems, the *payload* in charge of running complex algorithms to figure out the best behaviour of each car to avoid collisions while the second sub-system, the *wormhole* is running synchronous and robust algorithms aimed to check whether the payload timely sends corrections to the car actuators. In case a timely timing failure is detected, the wormhole can temporally take control of the car. As this technique is applicable to any domain where a safety-critical control is mandatory, we used a payload-and-wormhole-based architecture for the robot control layer.

Robot soccer game is an entertaining and well-known application of robot team cooperation. Actually, it shows many common points with our project like the need for all robots to real-time maintain a common view of the world. In the paper [4], the authors present an approach of view model which was successfully implemented during the 2002 RoboCup Sony competition. Due to some high latency in wireless communication, the robot team does not perform any view synchronization. In order to track a dynamic object like the ball, each robot combines local information from vision sensors with shared information sent by team-mates. The robot maintains timestamps and uncertainty values for each view object, uncertainty is updated when receiving new information and grows with time. Unlike in [4], our world model is based on a view synchronization but we use certainty flags associated with timestamps to give more or less weight to an object position in the world view.

3 Architecture overview

In this section, we give an architecture overview through the description of three key features of the middleware, depicted in figure 1. The left side shows the middleware layer division. The *wormhole* and *payload* are the middleware basic components. The *wormhole* is placed in cut-through configuration between the *payload* and the sensor/actuator layer and does not have access to the network device. The *payload* runs all complex tasks in charge of the robot control. All tasks are triggered by events broadcasted through a module tree. An example of module tree used in the *payload* is shown in the right side of the figure. Each group of modules is dedicated to a specific task: Position update, navigation or world view management. Here, we will focus on the view management and especially the synchronization mechanism.

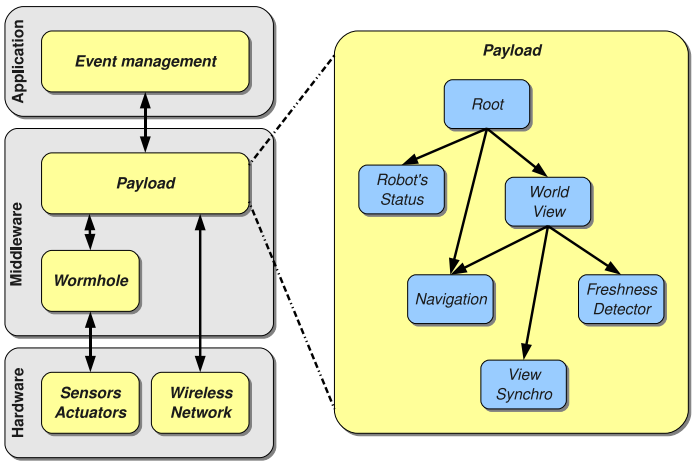


Fig. 1: Middleware layer division and module tree

3.1 Wormhole and payload model

The *payload*, may be asynchronous and runs complex and intelligent algorithms. These algorithms might not be deterministic and the computation time may vary especially if they require network communication. The wormhole is synchronous and runs simple and more robust algorithms. On the robot, some critical tasks like the collision avoidance require for the *payload* to timely send information to the hardware layer. In our case, this information will be new speed and heading commands.

The *wormhole*'s job is to check whether these commands are timely sent by the *payload*. If so, the *wormhole* will just forward them to the hardware layer. Otherwise, it will assume the robot's control by calculating and sending itself these commands. The *wormhole* uses a low-level navigation algorithm which only involves local information. The *wormhole* will keep the control until the *payload* starts again to send commands on time. If the *payload* does not regain stability or even no longer sends any command (the *payload* may have crashed), the *wormhole* can restart the whole *payload* process.

The *payload* sends the new commands in a structure called the *promise*. Each *promise* includes a deadline which enables the *wormhole* to control the *payload*'s timeliness. For each promise, the *payload* is expected to send the next command before the current deadline is exceeded.

3.2 Module definition

The event-based architecture is well adapted to robot management. It allows to design a flexible and modular architecture. Indeed, we can create an event type for any robot feature and dedicate a part of the tree to handle this event. Moreover, robot hardware is composed by sensors, actuators and communication

devices, each one of them can generate a special event or be triggered by this event. The *obstacle* event is an example of event which contains the distance values read from the local sensor.

The *payload* is composed by modules and groups of modules, all gathered in a tree. Each of them is in charge of managing a robot feature. A module can be seen as a process, with a short computation time, which is started by a single incoming event and can produce one or more outgoing events.

We will detail later the different types of events but basically, an event is broadcasted from a given location in the tree until the leaves. Each module which can consume this event, is started. That way, various modules can run at the same time.

3.3 Team management and view synchronization

Given that robots can move away from each other and go beyond their wireless range, a group can be split in various sub-groups. The unreliability of the wireless network can also lead to isolate a single robot if it temporarily loses the Wi-Fi signal. When two groups merge together, a synchronization mechanism is necessary to consolidate the information of each group view. The *payload* includes such a dynamic mechanism which ensures that all robot views are the same inside the group. The synchronization task as all other *payload*'s tasks is triggered by events. The *synchro* event will be detailed in chapter 4.3.

4 Design and implementation

We will now detail the payload and wormhole architecture, the implementation of the module structure and finally the world view synchronization algorithm.

4.1 Wormhole and Payload implementation

The *wormhole* relies on three modules as shown in figure 2: The *Timely Timing Failure Detection* (TTFD) monitors the timeliness of the asynchronous *payload* process and can activate the *Safety task* to assume control. The *Control task* receives the *promise* which includes the new speed and heading values and decides whether these values can be forwarded to the actuator layer. The *TTFD* sends as well control updates to the *payload* to inform it won back or lost the control. Ideally, the *payload* should use these control updates to improve its performance. In particular, it could try to real-time adjust the priority of some internal processes.

Logical flowcharts of the *TTFD* and *Control* tasks are given in figure 3. The *payload* runs in three modes: *"active"* when it has the control, *"disable"* when it loses the control after the latest deadline is exceeded and finally in *"test"*, when the *wormhole* receives a timely promise while the *payload* is disabled. The *test* mode is a transition period, the *wormhole* keeps the control and waits for the *payload* to meet the current deadline before giving him back control.

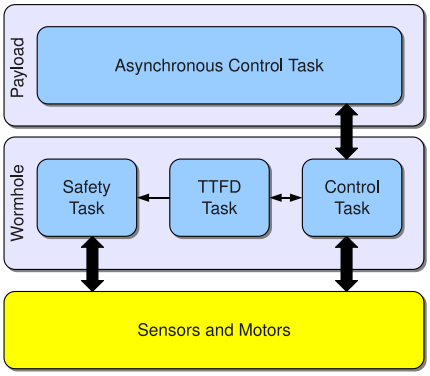


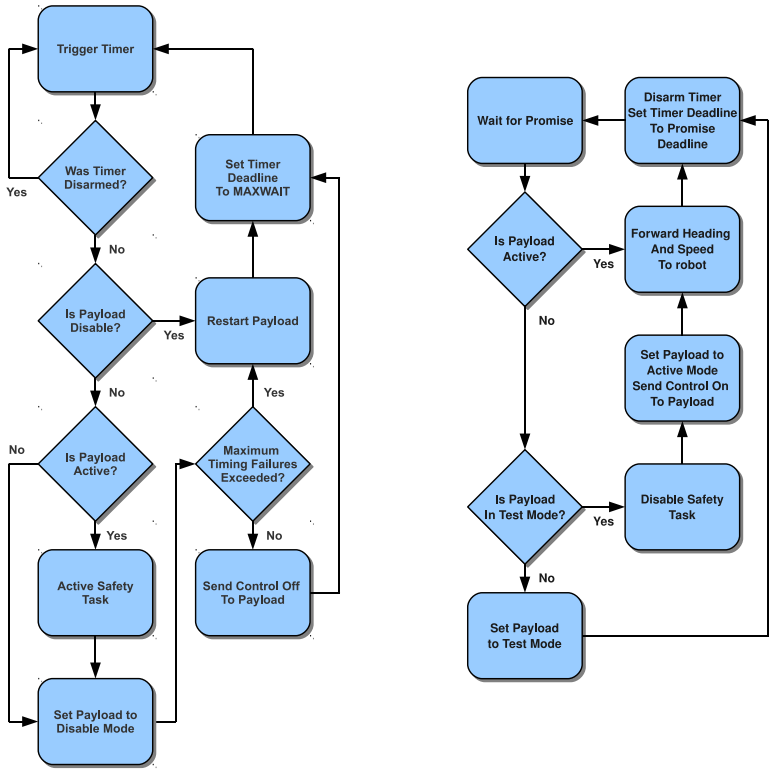
Fig. 2: Payload and Wormhole layers

There are two restart conditions for the *payload*. After setting it to *disable*, the *TTFD* task will increment a timing failure counter and will wait *MAXWAIT* milliseconds. If the failure counter is greater than a prior threshold or if no promise is received within this waiting period, the *TTFD* task will restart the *payload*.

4.2 Tree-oriented structure

The modules could be meshed as a graph and thus make event cycles possible. In order to avoid hazardous out-of-control cycles inside one robot or between several robots, we chose a tree-oriented module architecture. Each tree's branch has one or several parents. Events are top-down broadcasted until the leaves. A module computation is started if the current event can be consumed by the module. There are three types of events:

- **Hard events**: They are signals generated by the robot's hardware, e.g. the robot's clock (*beat* event) or a distance sensor measure (*obstacle* event). Such events are always broadcasted from the tree's root.
- **Local soft events**: These events are produced by a module and are broadcasted through the neighbour branches (modules with same parent) and the sub-branches. Any module can produce several soft events during the same computation.
- **Remote soft events**: Instead of being broadcasted locally, they are transmitted through the wireless network and sent to all other robots. Once delivered to a given robot, the event is broadcasted from the same branch as if it would be produced locally. This mechanism relies on two architecture properties: The module tree has the same structure for all robots which means that any path in the local tree matches the same path in a remote tree. Secondly, the path to locate the module which produced the event in the tree, is stored in the transmitted event. That way, we cannot have event



(a) TTFD task (b) Control task

Fig. 3: Wormhole logical flowchart

cycles between robots and the remote soft events meet the same constraints as the hard and local soft events.

Let's consider the module tree depicted in figure 4 with three robots in the same group. We assume that *e1* and *e2* are hard events, *e3* a local soft event and *e4* a remote soft event. Now, let's have a look on the started modules if *e1* is triggered on robot 1.

- Module 1 of Robot 1 (locally triggered by *e1*)
- Module 2 of Robot 1 (locally triggered by *e1*)
- Module 4 of Robot 1 (locally triggered by *e3*)
- Module 6 of Robot 2 (remotely triggered by *e4* with path *root.g1*.
- Module 6 of Robot 3 (remotely triggered by *e4* with path *root.g1*.

Although module 3 can consume the *e4* event, this module is not started in robots 2 and 3 because it cannot be reached from the path *root.g1*. What's

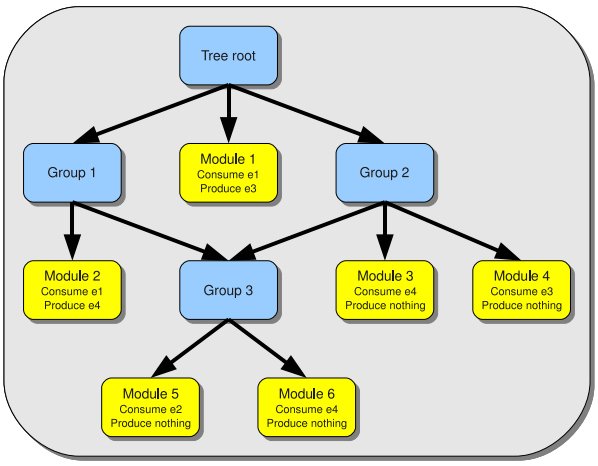


Fig. 4: Example of module tree common to robots 1, 2 and 3

more, this architecture opens the possibility of dynamically enabling or disabling a sub-branch of the module tree. When disabled, events are no longer broadcasted through this branch. The activation or disactivation can be performed by any computation module. In our project, each robot has three working modes: Wandering, Searching and Blocking (an intruder), each mode corresponds to a single branch of the navigation sub-tree. We use the branch enabling/disabling mechanism to activate the modules associated to the robot current mode.

4.3 World view synchronization algorithm

We will now describe in details the synchronization algorithm used to maintain in each robot a coherent world view when two or more groups are merging. This view is composed by all dynamic objects present in the world. The first part will deal with the algorithm principles and the second part will present some synchronization scenarios.

Algorithm overview: When two groups are merging, the synchronization is performed by exchanging a *synchro* event which contains the list of all view objects except for the robots position. This latter is already exchanged through the *hello* events so including this information in a *synchro* event would be redundant. The *synchro* event is normally sent by the group leader. The leader is the robot with the lowest id in the group. A group is identified by a single id. So all robots from a group share the same leader and group id. The *synchro* event reception is not centralized by the leader, each robot from the destination group, will handle the *synchro* event and extract the object list. The synchronization phase ends when all robots have the same leader and group id.

The basic steps below are associated to a faultless synchronization phase. By fault, we mean any event reception failure due for instance to a temporally

Wi-Fi signal loss. Different fault scenarios will be discussed in the next section. Two *synchro* events are exchanged during a faultless synchronization, the first *synchro* event is always generated by the group leader with the higher id.

- *Step 1*: Leader 1 receives a *hello* event from the group leader 2.
- *Step 2*: Leader 1 broadcasts a *synchro* event through the group 2.
- *Step 3*: Robots from group 2 receive the object list and update their view.
- *Step 4*: Leader 2 broadcasts a *synchro* event through the group 1.
- *Step 5*: Robots from group 1 receive the object list and update their view.

The algorithm 1 gives the modules involved in the synchronization phase. The *Hello Receiver* module (line 1) is triggered by an *object* event which is used by a robot to broadcast its own position, the module checks out whether this event comes from another leader. The *View Synchronizer* module (line 9) is triggered by a *synchro* event, it extracts the object list and updates the local world view (line 16). Finally, the *Freshness Detector* module (line 22) is triggered by the *beat* event and removes out-of-date objects from the robot's view. The *beat* is a hard event periodically generated by the system (see section 4.2).

In order to keep the algorithm clear, we won't detail below neither the *Hello Sender* module which is also triggered by the *beat* event and produces an *object* event, nor the *View Updater* module triggered by an *object* event and which updates the world view. The robot state parameters are as follows:

- *myId*: single robot identifier.
- *myView*: view objects including team-mate positions.
- *myLeader*: current group leader identifier.
- *myGroup*: current group identifier.

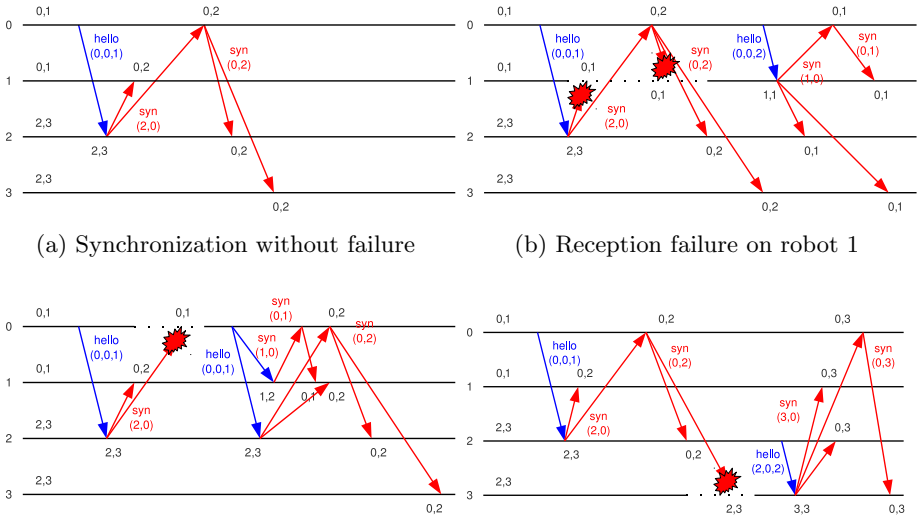
Examples of synchronization scenarios: Figures 5 and 6 show various synchronization scenarios with respectively two and three different groups merging at the same time. Each group is first composed by two robots, robots 0 and 1 for the first group, robots 2 and 3 for the second one and so on. Arrows identify events (blue for *hello* and red for *synchro* events) which are handled by robots for the synchronization phase. Other *hello* events broadcasted to periodically announce robot positions are not represented here. Each scenario is given as an example. Therefore, the number of events exchanged during a scenario could be different according to the order each event is delivered with. This statement is especially true if the number of groups merging at the same time is large.

In a robot time line, couple of black values correspond respectively to the leader and group id. The *hello* event parameters are the source robot, leader and group id. Finally, the *synchro* event parameters are the source and destination leader id (*leader1* and *leader2* in the algorithm 1).

Scenarios 5b, 5c and 5d highlight temporally reception failures which lead the robot to broadcast extra events to achieve the synchronization. Such failures could be due to a temporary Wi-Fi signal loss. The extra event phase is initialized by the faulty robot which receives a *hello* packet from its leader. This mechanism is implemented at line 5 of the algorithm 1.

Algorithm 1 View synchronization algorithm

```
1: upon event <object | type, id, leader, group> do
2:     if type = "robot" ∧ (leader ≠ myLeader ∨ group ≠ myGroup) then
3:         if id = leader ∧ myId = myLeader ∧ id < myId then
4:             synchronization(myLeader, leader)
5:         else if id = myLeader then
6:             synchronization(myId, leader)
7:             myLeader ← myId
8:
9: upon event <synchro | leader1, leader2, objList> do
10:    if myLeader = leader2 then
11:        if leader1 < myLeader then
12:            myLeader ← leader1
13:        if myId = myLeader then
14:            synchronization(myId, leader1)
15:        for all obj ∈ objList do
16:            trigger <object | obj.type, obj.id, obj.leader, obj.group>
17:        if leader1 > leader2 then
18:            myGroup ← leader1
19:        else
20:            myGroup ← leader2
21:
22: upon event <beat | > do
23:    updateRequired ← false
24:    for all obj ∈ view do
25:        if isUptodate(obj) = false then
26:            view ← myView − {obj}
27:            if obj.type = "robot" ∧ (obj.id = myLeader ∨ obj.id = myGroup) then
28:                updateRequired ← true
29:    if updateRequired = true then
30:        updateLeader()
31:
32: procedure SYNCHRONIZATION(leader1, leader2)
33:    objList ← {}
34:    for all obj ∈ myView do
35:        if obj.type ≠ "robot" then
36:            objList ← objList + {obj}
37:    trigger <synchro | leader1, leader2, objList>
38:
39: procedure UPDATELEADER
40:    myLeader ← myId
41:    myGroup ← myId
42:    for all obj ∈ myView do
43:        if obj.type = "robot" ∧ obj.id < myLeader then
44:            myLeader ← obj.id
45:        if obj.type = "robot" ∧ obj.id > myGroup then
46:            myGroup ← obj.id
```



(a) Synchronization without failure

(b) Reception failure on robot 1

(c) Reception failure on robot 0

(d) Reception failure on robot 3

Fig. 5: Examples of view synchronization between two groups

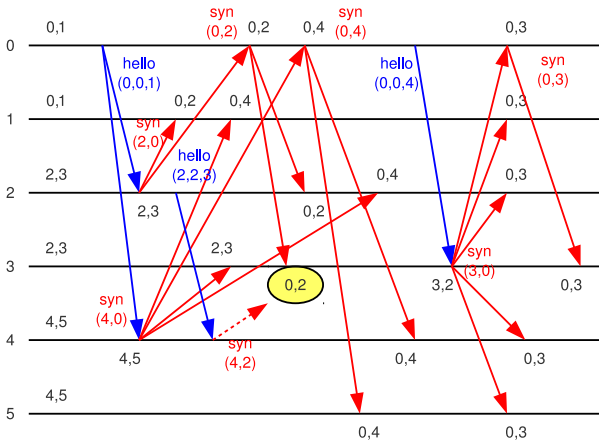


Fig. 6: Example of view synchronization between three groups

The last scenario 6, three groups merging at the same time, is unusual but shows the algorithm resilience. We can notice that at the end of the first "round", the robot 3 doesn't have the same group id than the others (yellow-circled value). The situation gets stable after the second *hello* event.

Advantages and drawbacks: This algorithm is very simple and offers resilience in signal loss situations. Nevertheless in some tricky scenarios, it could require more than one round, i.e. more than one *hello* event to stabilize itself. *Hello* events are periodically generated (according to the *beat* signal frequency), so increasing this beat signal frequency to accelerate the synchronization phase could be attractive but may on the other hand overload the wireless network and what's more, lead to some algorithm instability if this frequency is greater than half the mean round trip delay of the wireless network.

5 Conclusion and future work

We have proposed a middleware architecture aimed to offer a resilient control system for mobile robots. A middleware version was written in C and evaluated by means of robot simulation Java tools (Simbad v1.4). Most common scenarios like communication failures, robot group splitting and merging, or payload overload have been successfully tested. The next step is now to port the middleware to an embedded platform based on an ARM chip and a FPGA.

References

1. Luís Marques, António Casimiro, Mário Calha, Design and development of a proof-of-concept platooning application using the HIDENETS architecture, DSN '09: Proceedings of the International Conference on Dependable Systems and Networks, 223-228, 2009.
2. Casimiro, A. and Rufino, J. and Marques, L. and Calha, M. and Veríssimo, P., Applying architectural hybridization in networked embedded systems, The Seventh IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, November, 2009.
3. Paulo Veríssimo. Travelling through wormholes: a new look at distributed systems models. SIGACT News, 37(1):66–81, 2006.
4. Maayan Roth, Douglas Vail, and Maria Manuela Veloso, A Real-time World Model for Multi-Robot Teams with High-Latency Communication. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), October, 2003.
5. M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. IEEE Transactions on Robotics and Automation, 17(3):229–241, 2001.
6. J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. IEEE Transactions on Robotics and Automation, 7(3):376–382, 1991.
7. Andrew Howard, Maja J Matarić and Gaurav S Sukhatme. Localization for Mobile Robot Teams Using Maximum Likelihood Estimation. IEEE/RSJ International Conference of Robotics and Intelligent Systems (IROS): 434-459, 2002.
8. Kristel Verbiest, Eric Colon. Securing Hostile Terrain with a Robot Team. Fourth International Workshop on Robotics for risky interventions and Environmental Surveillance-Maintenance, RISE'2010– Sheffield, UK, January 2010
9. Rodney A. Brooks, A Robust Layered Control System For a Mobile Robot, Massachusetts Institute of Technology, Cambridge, MA, 1985.

Using the MegaBlock to Partition and Optimize Programs for Embedded Systems at Runtime

João Bispo[1], João M. P. Cardoso[2],

[1] IST/Universidade Técnica de Lisboa, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
joaobispo@gmail.com,
[2] Universidade do Porto, Faculdade de Engenharia, Departamento de Engenharia Informática, Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
jmpc@acm.org

Abstract. This paper presents our recent research efforts addressing the dynamic mapping of sections of execution to a coarse-grained reconfigurable array (CGRA) coupled to a General Purpose Processor (GPP). We are considering the common scenario of a GPP – a RISC processor – using the CGRA as a co-processor to speedup applications. We present a partitioning scheme based on large traces of instructions (named Megablock). We show estimations of the speedups achieved by considering the Megablock.

Resumo. Este artigo apresenta os nossos esforços mais recentes em relação ao mapeamento dinâmico de secções de programas a correr em processadores de âmbito geral (GPPs) para agregados reconfiguráveis de grão grosso (CGRAs). Na abordagem actual consideramos um cenário em que temos um GPP – processador RISC – que utiliza um CGRA como co-processador para acelerar aplicações. Apresentamos um método de particionamento baseado em grandes blocos de instruções (denominados MegaBlocos) e mostramos valores estimados de acelerações do tempo de execução quando se considera o MegaBloco como unidade de partição.

Keywords: Reconfigurable Computing, Dynamic Mapping, Just-In-Time Compilation, Binary Translation.

1 Introduction

The execution of applications on general purpose processors (GPPs) can be enhanced – e.g., lower execution time, lower energy consumption – by moving computationally intensive parts (hot-spots) to specialized custom hardware components such as Reconfigurable Processing Units (RPUs) [1, 2]. This is becoming common practice in high-performance embedded systems. It is common to use a programmable processor, often a RISC-like GPP, to run the application and use a custom hardware coprocessor (e.g., CGRAs – Coarse-Grained Reconfigurable Arrays) when certain requirements cannot be met by the GPP alone.

However, to be able to use the custom hardware, we must rewrite part of the application and explicitly call this hardware when needed. This can be accomplished

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 699–710

by several means (e.g., manually by a programmer, automatically by a compiler). By using techniques from both binary translation and dynamic compilation, it is possible to translate the application and insert calls to the hardware at runtime. This way we can transparently move critical sections of programs running on a GPP to a CGRA co-processor without pre-changing the program binaries. We refer to this as dynamic mapping. It has already been successfully applied [3], but research efforts about the benefits and the feasibility of dynamically partitioning binaries to reconfigurable architectures are relatively recent [4].

This paper shows our most recent efforts on dynamic mapping. We identify a set of characteristics that when present in the critical sections can benefit dynamic mapping, and we propose a novel partitioning method which can extract blocks of instructions [5] with those characteristics in mind. We show how this partitioning method can impact performance.

This paper is organized as follows. Section 2 introduces the dynamic mapping problem and motivation. In Section 3 we explain our approach to dynamic mapping and we propose the MegaBlock partitioning method. Section 4 presents experimental results regarding our approach and Section 5 introduces related work. Finally, Section 6 concludes the paper.

2 Dynamic Mapping

As previous work has shown, if we move the critical loops of a program to dedicated hardware units, we can have significant performance improvements [6]. There have been many proposals on accelerators for reconfigurable computing, as well as a plethora of architectures [7, 8]. Most well-known examples include Adres [9], Morphosys [10], Chimaera [11], and XPP [12]. Each one of these architectures proposes unique features and tries to address faster execution and/or energy savings for a set of algorithms. Currently, there is a wide choice of hardware accelerators and fine-grained reconfigurable fabrics such as FPGAs (Filed-Programmable Gate Arrays) are a fairly cheap technology to implement them. The main obstacle to custom hardware units is the significant cost of rewriting the programs to take advantage of those units.

A common approach has been to develop tools which automatically partition a program (typically in C) into software and hardware parts [13, 14]. With the help of profiling information, the tools detect small sections of code where the program spends most of its time (critical kernels or hot-spots). This approach is applied at compile time (statically). Since it is static, it can use more complex algorithms than dynamic approaches. On the other hand, the binary generated by the tools is often tied to a very specific setup. Even when the tool supports several families of the same architecture (e.g., with variations in the number of functional units), at compilation time the options usually are compiling to a very specific architecture, or to the lowest common denominator. In addition, if the execution of a program is sensitive to changes in the input data, the information collected during profiling might not hold between executions, limiting the adaptability of the generated binary.

During static partitioning, we can only move parts of an application to hardware if we have access to its code. In this approach, pre-compiled libraries (e.g., DLLs) are usually out of the partitioning scope, and consequently they are not considered for target-specific compiler optimizations.

Dynamic mapping can make RPUs transparent without compromising existent binary portability, expose more optimization opportunities and expand the use of reconfigurable hardware in embedded computing systems. However, dynamic mapping represents a difficult challenge, since it implies we need to execute many of the tasks performed by static partition at runtime. On the other hand, it provides access to information previously not available, which can be used for further optimizations.

We are considering the common scenario of a GPP using a co-processor to speedup applications. In such a case, the execution will be switching back and forth between the GPP and the co-processor.

We focus our work on the level of the instructions executed by the GPP. By working at a higher level (e.g., doing the partitioning of the program on C code) we might not have access to important information about the execution flow of the program.

3 Our Approach to Dynamic Mapping

The main objective of our work is to contribute in bridging the gap between software and reconfigurable hardware. Embedded computing is a good target since it is an area where it is common to find systems including customized hardware modules and reconfigurable hardware.

We want to move parts of programs to hardware to improve one or more particular aspects (e.g., execution time, energy consumption). So, instead of starting with the hardware and propose a specific architecture, in our approach we want to start with the programs, and discover what kind of opportunities they have for dynamic mapping.

Nonetheless, the particular mapping techniques will depend on the target co-processor architecture, memory interface, and available communication. We think that to maximize the impact of dynamic mapping, we should go beyond the Basic Block and be prepared to map blocks of instructions with dozens to hundreds of instructions. Bearing this in mind, we choose to base our work on the general architecture shown in Figure 1. This kind of target architectures with a RISC-like GPP is commonly used in embedded systems. Currently, we use the Xilinx MicroBlaze *softcore* processor [15] as the GPP to run the programs. Dynamic mapping could possibly be applied to other types of hardware co-processors, but we choose to focus our work on CGRAs, since they generally need less mapping efforts than finer-grained alternatives (e.g., FPGAs). Note, however, that this does not constrain the use of FPGAs as CGRAs can be mapped to the FPGA hardware resources, which is a trend in the reconfigurable computing area.

We present in this paper a novel approach to one of the challenges in dynamic mapping: identifying what portions of code should be mapped (partitioning).

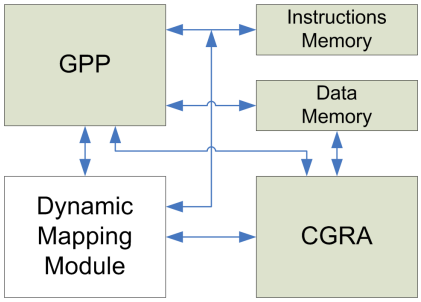


Figure 1. General Architecture

Regarding the architecture assumptions previously described in this section, we identified three issues.

Firstly, the longer a segment of code executes in the co-processor uninterruptedly, the higher the impact of dynamic mapping. This will reduce the communication and reconfiguration overhead [13], and since each partition will incur in a mapping cost the first time it is found and a reconfiguration cost every time it is used, it is desirable to find mapping candidates which will have a large number of iterations.

Secondly, as stated by Amdahl's Law, we need to move a large portion of the program execution to the co-processor if we want to have a significant impact. E.g., for a speedup of 2×, we will always need to move more than 50% of the execution from the GPP to the co-processor.

Lastly, branches are a common occurrence in code running on GPPs, and do not translate well to the usually highly parallel, data-flow co-processors. Branches can also prevent several optimizations and limit the amount of Instruction-Level Parallelism (ILP). Hardware accelerators work best when the control-flow is very low or non-existent.

Taking these issues into account, we consider that a good candidate for mapping would be a segment of branchless code (control-flow issue) which repeats itself a high number of times during execution (iteration issue). It is important that such segments represent a significant portion of the program execution (coverage issue). Figure 2 illustrates the segments we are currently identifying in an execution trace, in an example in pseudo-code.

The BasicBlock is formed by a sequence of instructions with single entry-point and single exit-point – basic blocks end when a branch or jump instruction appears.

A similar, yet more powerful type of segment is the SuperBlock. SuperBlocks are regions of code with single entry-point and multiple exit-points. Originally, it was proposed as a technique to extract more ILP from static compilation [16], but it was later adapted for dynamic compilation [17]. The dynamic version of the SuperBlock represents a common, biased path along several BasicBlocks. A SuperBlock is built by adding BasicBlocks until we reach a BasicBlock that ends with a backward jump. The jump starts a new SuperBlock.

Expanding on the idea of the SuperBlock, we propose another type of segment, the MegaBlock, as a sequence of SuperBlocks, with a bias towards consecutive repetitions. A MegaBlock is built by identifying a sequence of SuperBlocks, up to a

predetermined size. When a sequence of SuperBlocks repeats itself at least one time, that sequence is considered as a MegaBlock with multiple iterations. SuperBlocks which do not form repeatable sequences are also considered as MegaBlocks, albeit with only one iteration. It should be noted that when these three kinds of segments are considered individually, they only have one execution path and the only control-flow inside the blocks are side-exits.

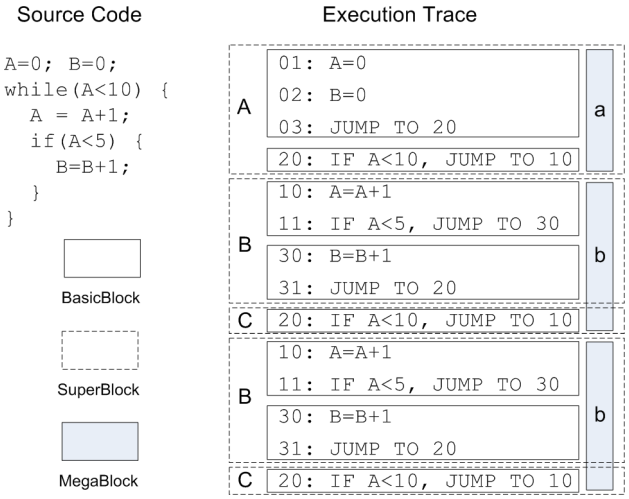


Figure 2. Program execution partitioning according to BasicBlocks, SuperBlocks and MegaBlocks

To use these segments in dynamic mapping, it is important that we can detect and extract them during runtime. To detect BasicBlocks, we identify branch instructions. SuperBlocks can be detected by identifying backward branch instructions. To find MegaBlocks at runtime we propose a technique which first, uniquely indentifies SuperBlocks by using the first addresses of their BasicBlocks to create a hash value (e.g., [16]). Using a hardware pattern matching module, we can efficiently find MegaBlocks within a stream of SuperBlock hashes. We could make the detection of MegaBlocks over the BasicBlocks instead of the SuperBlocks, but the coarser granularity of the SuperBlock reduces the pattern matching requirements significantly.

Before mapping a section of the program execution, we can apply optimizations to expose more ILP, or to reduce the number of instructions to map. Although this is not explored in this paper, we will refer some considerations about these optimizations. Since the algorithms should perform during runtime, we favor algorithms which map well to hardware. We focus on algorithms which can be applied to a stream of instructions and which use tables to store temporary data, (instead of, e.g., graph representations). The reason is that, later they might be easier to translate to hardware. We use a simplified Single-Static Assignment (SSA) format without Phi functions [18], and maintaining the original number of the registers added with the number of each specific definition. It is simpler as the algorithms are applied over blocks of

instructions with a single execution path. Since there is no more than one path at any given moment, a use can only be reached by a single definition – which can be kept in a table – removing the need for a Phi function.

4 Experimental Results

We chose a set of 13 benchmarks which are commonly used in embedded computing and which represent a wide range of integer computations. The benchmarks used are: adpcm coder and decoder, autocorrelation, bubble sort, discrete cosine transform, dot product, fdct, fibonacci, fir, max, pop_cnt, sobel and vecsum. All benchmarks were compiled with *mb-gcc* (the GCC compiler targeting MicroBlaze) using different levels of optimization. The number of instructions executed for the benchmarks range from around 500 instructions to 300,000 instructions.

The Megablock identification uses the maximum size of the sequence of SuperBlocks as a parameter. Figure 3 represents the coverage of MegaBlock based partitioning with different maximum sequence sizes. Table 1 shows the sizes, in number of executed MicroBlaze instructions of the corresponding MegaBlocks.

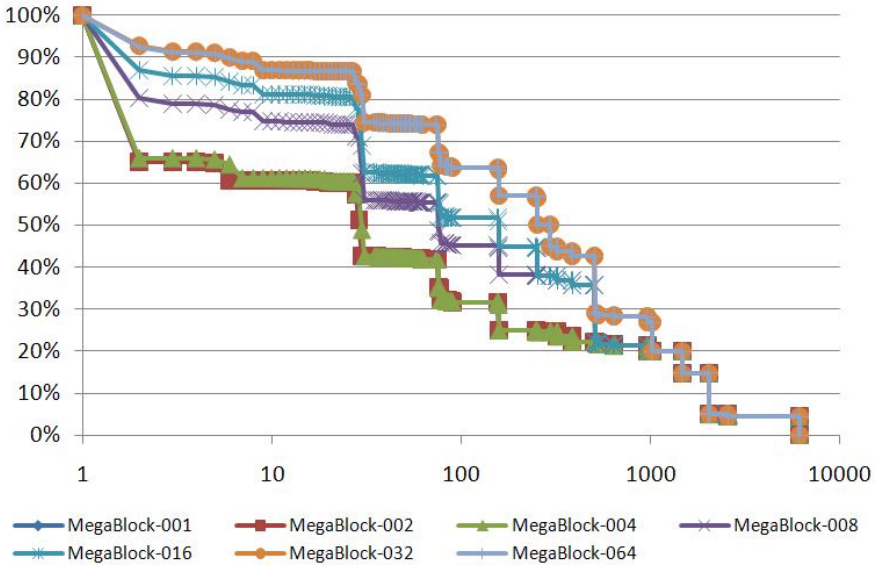


Figure 3. Portions of the program execution (*Y axis*) that are covered by MegaBlocks which have *at least* a certain amount of iterations (*X axis*), according to the maximum number of SuperBlocks a MegaBlock can have. Since every block has, at least, one iteration, value 1 on the *X axis* corresponds to 100% of the program execution on the *Y axis*

Figure 3 and Table 1 indicate that the higher the maximum sequence size, the more coverage we have, but the bigger will be the MegaBlocks. For sequence sizes above 32 there is no coverage gain for the presented benchmarks. There is a significant

difference in the average size of MegaBlocks between a maximum sequence size of 16 and 32, but the average size in the latter case is still comfortably inside the usual size of blocks in approaches which implement small loops [6]. The coverage is an average for all benchmarks, when executing the binaries previously generated with the O3 flag. Regarding the three partitioning schemes previously presented, the results indicate that the MegaBlock with parameter 32 (i.e., considering MegaBlocks with up to 32 SuperBlocks) has the potential to represent a significant portion of program execution. For the considered benchmarks, on average, MegaBlocks with 10 or more iterations represent almost 90% of the total execution; MegaBlocks with 100 or more iterations represent more than 60% of the total execution. Since the MegaBlock has a single execution path, mapping this block to hardware may need less effort than mapping blocks with more complex control-flow.

Table 1. MegaBlock sizes with respect to the number of executed instructions. We present results when we only take into account MegaBlocks with 2 or more iterations, and MegaBlocks with 10 or more iterations. Note that we are not interested in mapping blocks which have only one iteration. The weighted average has into account the number of iterations each MegaBlock of a particular size has

Max Sequence Size	Minimum Size		Maximum Size		Weighted Average	
	2	10	2	10	2	10
1	4	4	106	106	7.5	7.4
2	4	4	106	106	7.5	7.4
4	4	4	106	106	7.5	7.4
8	4	4	783	106	9.0	8.8
16	4	4	783	309	10.4	10.3
32	4	4	783	309	26.0	25.6
64	4	4	783	309	26.0	25.6

Figure 4 presents a comparison between several partitioning methods using the 13 benchmarks. The execution of the programs was partitioned in blocks, and we measured, during the program execution, the number of consecutive iterations that occurred for each block. Besides the methods presented in Figure 2, we also implemented the partitioning method used by the Warp processor [6]. Note, however, that this last partitioning method detects complete loops with control-flow, while the others are biased, branchless paths. The curves in the figure should be seen relative to one another: they are particular for a set of benchmarks, and even the same program can present a different number of iterations with a different set of input data. For the considered benchmarks, the MegaBlock with a maximum pattern size of 32 is consistently above the other considered methods. This means that for the same number of iterations, the blocks found by this partitioning method represent a higher percentage of the executed code. It was expected that the Warp partitioning method could present a higher coverage, since it is covering not only the frequent path of the loop but also all the other paths of the loop. It seems that the method used by Warp [6] detects only inner loops. As the MegaBlock detects patterns of SuperBlocks, if an inner loop can fit in a small number of SuperBlocks and the size of the maximum sequence is sufficiently big, the MegaBlock partitioning method will automatically

consider small unrolled inner loops. The SuperBlock partitioning method follows the Warp partitioning method very closely. This is to be expected, since them both use backward branches to detect small loops.

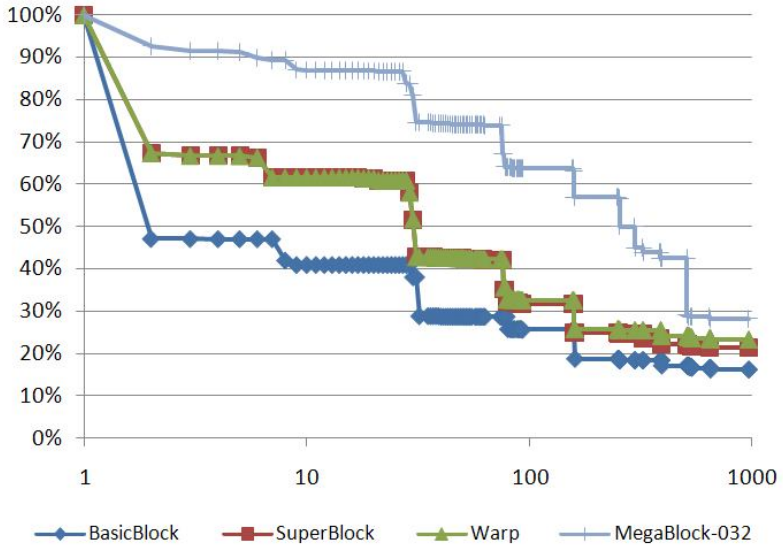


Figure 4. Portions of the program execution (*Y axis*) that are covered by blocks, identified by several partitioning techniques, which have *at least* a certain amount of iterations (*X axis*). We are using MegaBlocks which have, at most, 32 SuperBlocks

To analyze the potential speedups that can be obtained using our approach, we considered a hypothetical large 2D CGRA architecture coupled to the MicroBlaze. This CGRA consists of a number of rows, each one with functional units (FUs) and a single load/store unit. Each row is executed in one clock cycle. We consider that each instruction in the program execution can be mapped to a functional unit. We imposed a communication restriction, where FUs from a given row could only communicate with the FUs of the row immediately below. When an FU needed data from another FU from a distance higher than one row, the data items are communicated through other unoccupied FUs using "move" instructions. The only exceptions were data inputs, which can be read by any row. We also imposed restrictions for the memory operations: at any given row, there is only one load/store operation. This architecture is very similar in concept to the DIM architecture [19].

The mapping algorithm is based on the algorithm used by Clark *et al.* to map instructions to the CCA [20]. The MegaBlock is read as a stream of instructions, and each incoming instruction is placed on the first row which respects the data dependencies. After placement of the instruction, the algorithm checks if the instruction can receive the required data, and if not, inserts the necessary 'move' instructions. Additionally, it uses a conservative approach for memory instructions, mapping any load operation after the last store operation and respecting the occurrence order of store operations and possible side-exits. The speedup figures

account for communication overheads between reconfigurations, and assume we need one clock-cycle to communicate each live-in and live-out register.

Figure 5 shows the speedups for each benchmark across several levels of compiler optimization, when we use the MegaBlock partitioning method with a maximum sequence size of 32 and we move to hardware all blocks which have at least 10 iterations. As shown in Figure 5, we can achieve speedups from 2× to 4× on average, depending on the optimization level of the compiler. Higher optimization levels show higher speedups across most benchmarks, which come from a higher coverage rate for those optimization levels. This might happen because higher optimization levels can represent code in a more efficient format, which can benefit the pattern matching (e.g., less SuperBlocks for a given pattern). When there are simultaneously patterns of several sizes (e.g., the sequence AAAA has patterns of size one – A – and two – AA), since the pattern matching algorithm gives priority to the pattern with the smallest size, it is able to extract the smallest common kernel, even when the compiler uses optimization techniques which increase the size of the code (e.g., loop unrolling).

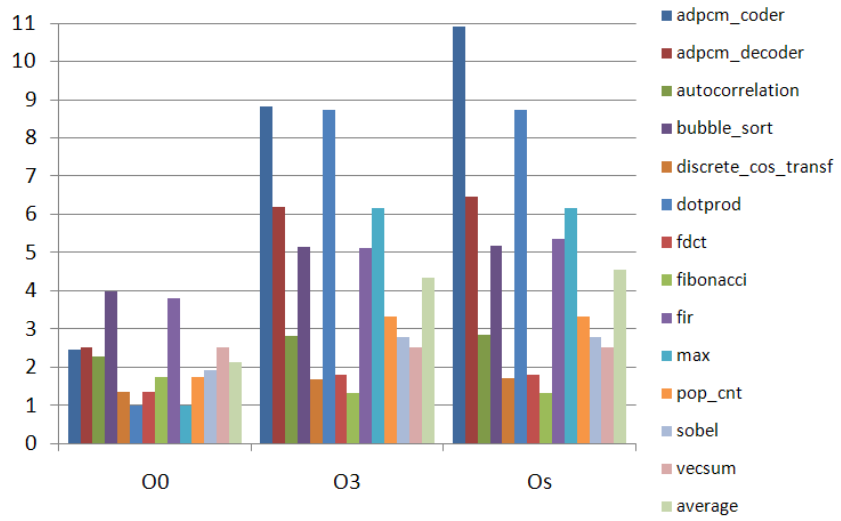


Figure 5. Speedups across different levels of compiler optimizations

5 Related Work

In the context of embedded systems, there have been several efforts addressing the dynamic mapping applications to RPUs.

Lysecky et al. [6] propose the Warp Processor, a system which includes a GPP, a fine-grained RPU and a dynamic mapping module. The dynamic mapping module automatically detects critical loops executing on the GPP and maps the corresponding binary code to the fine-grained RPU. In a posterior work [21], they use the same technique to improve the performance of a MicroBlaze *softcore* processor.

The Configurable Compute Accelerator (CCA) [22] is a special-purpose unit designed to be integrated in the pipeline of a GPP and executes a restrict set of Data-Flow Graphs (DFGs). The CCA cannot be directly accessed through programming, and instead, the unit itself has hardware support for binary translation, which automatically moves code from the instruction pipeline to the CCA.

Beck et al. [19] propose the Dynamic Instruction Merging (DIM) technique, a binary translation method to transparently map Basic Blocks from a general purpose MIPS processor to a coarse-grained reconfigurable array. They tightly couple the coarse-grained array to the processor, working as an additional functional unit in the execution stage of the pipeline. The objective of this architecture is to accelerate embedded systems that need to execute many different kinds of tasks.

Regarding these three approaches, Warp uses fine-grained reconfigurable hardware as the target RPU of dynamic mapping. Comparing to a coarse-grained RPU, it trades-off higher flexibility in the circuitry that can be implemented with higher mapping overhead. It is also an approach which needs a greater mapping effort, and that is not tightly coupled to the processor: both the CCA and the DIM are integrated in the pipeline of the processor, while the Warp RPU works as a co-processor. In the other hand, this enables the mapping of larger blocks in the Warp Processor. It implements complete loops, while the CCA and the DIM exploit ILP inside a small number of Basic Blocks.

6 Conclusions

This paper presented our approach to dynamically migrate computationally intensive sections of program execution from a general purpose processor to a coarse-grained reconfigurable array working as a co-processor. We proposed the MegaBlock for partitioning and presented experimental results showing a comparison between our approach and other common approaches such as the BasicBlock and the SuperBlock.

Ongoing work is addressing runtime optimizations and studying their impact in the final speedups. Future work will address hardware implementations of some of the modules needed to implement our dynamic mapping approach in order to quantify some of the resultant characteristics.

Acknowledgments

This research has been sponsored by the Portuguese Science Foundation (FCT) under research grants PTDC/EEA-ELC/70272/2006 and SFRH/BD/36735/2007.

References

[1] Z. Guo, W. Najjar, F. Vahid, and K. Vissers, "A quantitative analysis of the speedup factors of FPGAs over processors," in *FPGA '04: Proceedings of*

the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, Monterey, California, USA, 2004, pp. 162-170.

[2] J. Henkel, "A low power hardware/software partitioning approach for core-based embedded systems," in *Annual ACM IEEE Design Automation Conference: Proceedings of the 36 th ACM/IEEE conference on Design automation*: Association for Computing Machinery, Inc, One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA, 1999.

[3] J. C. Dehnert, B. K. Grant, J. P. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson, "The Transmeta Code Morphing™ Software: using speculation, recovery, and adaptive retranslation to address real-life challenges," in *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*: IEEE Computer Society Washington, DC, USA, 2003, pp. 15-24.

[4] G. Stitt, R. Lysecky, and F. Vahid, "Dynamic hardware/software partitioning: a first approach," in *Proceedings of the 40th conference on Design automation*: ACM New York, NY, USA, 2003, pp. 250-255.

[5] J. Bispo and J. M. P. Cardoso, "On Identifying Segments of Traces for Dynamic Compilation," in *20th International Conference on Field Programmable Logic and Applications (FPL'10)*, PhD Forum, Milano, Italy, 2010. (accepted)

[6] R. Lysecky, G. Stitt, and F. Vahid, "Warp Processors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, pp. 659-681, 2006.

[7] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*: Morgan Kaufmann/Elsevier, 2008.

[8] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in *Proceedings of the conference on Design, automation and test in Europe*: IEEE Press Piscataway, NJ, USA, 2001, pp. 642-649.

[9] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," in *Field-Programmable Logic and Applications*, 2003, pp. 61-70.

[10] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *Computers, IEEE Transactions on*, vol. 49, pp. 465-481, 2000.

[11] Z. A. Ye, A. Moshovos, S. Hauck', and P. Banerjee, "CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, 2000, pp. 225-235.

[12] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, "PACT XPP—A Self-Reconfigurable Data Processing Architecture," *The Journal of Supercomputing*, vol. 26, pp. 167-184, 2003.

[13] K. Bondalapati, P. Diniz, P. Duncan, J. Granacki, M. Hall, R. Jain, and H. Ziegler, "DEFACTO: A design environment for adaptive computing technology," 1999, pp. 570-578.

[14] Y. Yankova, G. Kuzmanov, K. Bertels, G. Gaydadjiev, Y. Lu, and S. Vassiliadis, "DWARV: DelftWorkbench Automated Reconfigurable VHDL

Generator," in *VHDL generator", the 17th International Conference on Field Programmable Logic and Applications (FPL'07*: Citeseer, 2007, pp. 697-701.

[15] I. Xilinx, "Microblaze processor reference guide," *reference manual*, 2006.

[16] W. M. W. Hwu, S. A. Mahlke, W. Y. Chen, P. P. Chang, N. J. Warter, R. A. Bringmann, R. G. Ouellette, R. E. Hank, T. Kiyohara, and G. E. Haab, "The superblock: an effective technique for VLIW and superscalar compilation," *The Journal of Supercomputing*, vol. 7, pp. 229-248, 1993.

[17] V. Bala, E. Duesterwald, and S. Banerjia, "Dynamo: a transparent dynamic optimization system," in *Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation* Vancouver, British Columbia, Canada: ACM, 2000.

[18] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck, "An efficient method of computing static single assignment form," in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* Austin, Texas, United States: ACM, 1989.

[19] A. C. S. Beck, M. B. Rutzig, G. Gaydadjiev, and L. Carro, "Transparent reconfigurable acceleration for heterogeneous embedded applications," in *Proceedings of the conference on Design, automation and test in Europe* Munich, Germany: ACM, 2008.

[20] N. Clark, J. Blome, M. Chu, S. Mahlke, S. Biles, and K. Flautner, "An Architecture Framework for Transparent Instruction Set Customization in Embedded Processors," in *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, 2005, pp. 272-283.

[21] R. Lysecky and F. Vahid, "Design and implementation of a MicroBlaze-based warp processor," *ACM Transactions on Embedded Computing Systems*, vol. 8, 2009.

[22] N. Clark, M. Kudlur, H. Park, S. Mahlke, and K. Flautner, "Application-Specific Processing on a General-Purpose Core via Transparent Instruction Set Customization," in *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture* Portland, Oregon: IEEE Computer Society, 2004.

A Framework for QoS-Aware Service-based Mobile Systems

Joel Gonçalves, Luis Lino Ferreira

CISTER Research Center
Polytechnic Institute of Porto (ISEP/IPP)
{vjmg; llf@isep.ipp.pt}

Abstract. In this paper we propose a framework for the support of mobile application with Quality of Service (QoS) requirements, such as voice or video, capable of supporting distributed, migration-capable, QoS-enabled applications on top of the Android Operating system.

Keywords: Quality of Service, mobile systems, Android OS.

1 Introduction

Mobile applications are increasingly more ubiquitous and more dynamic. Furthermore, mobile systems are increasingly open to third-party developed applications, being expectable that users put more and more pressure on locally available resources. But, even considering the substantial increase in devices' capabilities, it is not expectable that they will be able to simultaneously support all applications the users may want to execute. The solution to this is to allow applications to scavenge resources available in neighbor nodes by allowing applications to migrate some or all of its services into other nodes (or from other nodes). Therefore, there is a growing need to develop frameworks and applications that are able to reconfigure considering system-wide distribution of application and resources. Applications must therefore support: i) the capability to connect seamlessly to remote services and; ii) the capability to move some of its services to remote nodes.

Our goal is to provide augmented functionalities (quasi-)transparently in existent middleware and operating system. The Android operating system is used both due to its open source nature and potential market, but also due to its innovative architecture. Although its use to support real-time applications is still debatable [4], it nevertheless provides a suitable architecture for QoS-aware applications in ubiquitous, embedded systems.

Android applications (constituted by Intents, Activities and Services) and its resources are contained in an Android Package (*APK*) which can be installed in a local device. During runtime, the APKs components can be assembled to create dynamic applications, but these functionalities are only available in the local node. In this work, we extend this concept to a fully distributed and dynamic environment, where applications use the Activities and Services from the *APKs* whether they are installed locally or remote. Further, we allow *APKs* to migrate to other nodes, by user demand or due to system's reconfiguration. Finally, the framework is also able to support

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 711–714

applications with QoS requirements, both during its run-time operation and particularly during the migration of services.

The *APK*s are executed as independent processes, with distinct permissions, and, importantly, also with different QoS parameters. Figure 1 provides an example: the initial scenario is presented in the left part, where Device 1 is executing an application that uses services from three distinct *APKs*, in Device 1 (*APK* A) and Device 2 (*APK* B and *APK* C).

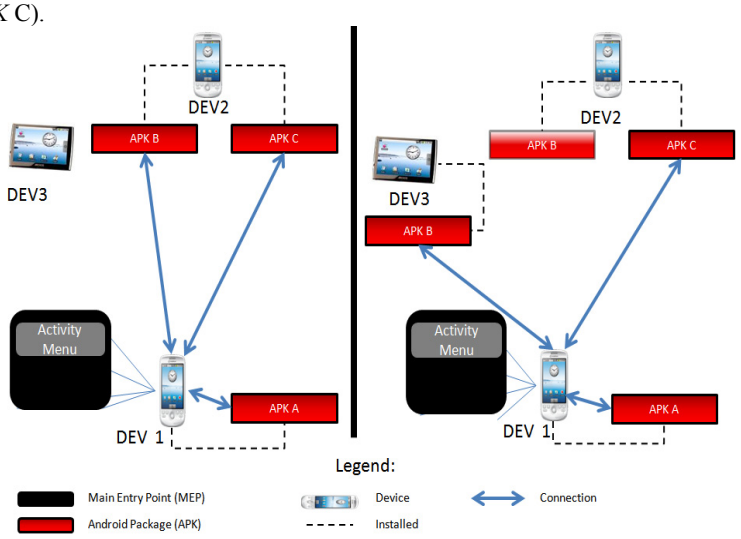


Fig. 1. Example scenario

The right side of Figure 1 provides a new system configuration, which includes a new device: Device 3. Assuming that this device has more resources available than Device 1, it enables a new application configuration with a higher QoS level if *APK* B is migrated to Device 3.

Algorithms and frameworks as the ones proposed in CooperatES [3] can be used to find the new configuration for the system services, maximizing the rewards for the overall system. As a result of the evaluation, Device 3 now offers the services of *APK* B. Mobile code mechanisms of the proposed framework support such approach, making possible to transfer (guaranteeing the QoS requirements), the code and state of service B from Device 1 to Device 3, install the corresponding *APK* file, rebind the connections between Device 1 and the service, and continue its operation.

2 Framework Architecture Overview

In order to implement our approach we assume the existence of a QoS Manager on the Android Operating System, the addition of application layer features to support the core functionalities of a mobile code framework and the use of supporting libraries which allow programmers to use the full set of capabilities offered by the framework.

The *Framework Core* functionalities (Figure 2) is constituted by separate modules implemented as an Android services, which takes care of service migration to and from another node, interacting with the QoS Manager in order to determine if the QoS requirements of the service can be supported.

Any Android application can use the services of the framework in order to support installing and running an Android *APK* in another node. More advanced services, which require the rebinding of connections between components, are only supported if applications use the *Mobile Library*.

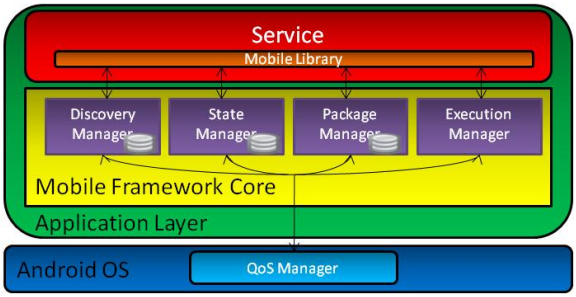


Fig. 2. Framework core modules

The *Mobile Library* offers a set of helper classes that extend the core functionalities of Android with QoS and distributed capabilities. The library allows to transparently extend the *Activity* and *Service* abstractions for distributed systems. Consequently, application code is only required to send an *Intent*, after which the framework is responsible for determining if it refers to a local or remote component. The library also implements the services required for communication establishment and automatic rebinding of service's connections when a service migrates. This library [5] already implements some generic mechanisms, which can be immediately used by developers, but also allows extensions to the base classes for new implementations.

The core services provided by the framework are: *Discovery Manager*, *Package Manager*, *State Manager* and *Execution Manager*.

The *Discovery Manager* module is designed to discover neighbour devices on a local network and advertise the host device available resources. To that purpose, every node in the network periodically broadcasts information regarding its status and installed services, such as the *APKs* installed, their associated Intents interfaces and resource availability.

The *Package Manager* is used to install, uninstall and transfer the code of *APKs* between Android devices. Applications can start executing when the node receives an Intent request from a remote *Execution Manager* (see below). It is also responsible for the interaction with the *QoS Manager* in order to request specific QoS levels for the service being handled. It is the responsibility of the *QoS Manager* to accept or reject service installations if the QoS required level cannot be guaranteed.

The *State Manager* handles transfer of state for statefull services. The *State Manager* is responsible for transferring either the full state or specific state items similarly to what has been proposed in [1, 2]. The flexibility on the implementation of the state migration policies can be of paramount importance for the use of code

mobility techniques in real-time systems since it can allow the reduction on the unavailability time of a service.

The *Execution Manager* allows launching services on a host device or on a remote node. Android *Intents* are exchanged between devices and used locally to start up a service, which can be a standard Android *Activity* or *Service. Intents* that address *Activities* or *Services* in remote nodes are parsed to extract their parameters and sent to the remote *Execution Manager*, which then reconstructs the intent and launches it locally (on the remote node).

Services which are based on the framework use the new versions of *Service* and *Activity* classes; therefore, when calling for a remote service they can connect directly using the functionalities provided by the Mobile Library.3

3 Conclusions

In this paper we proposed a framework for the development of distributed QoS aware applications with self-reconfiguration capabilities. The framework is particularly targeted for the Android Operating System and its implementation extends the main abstractions used in Android – *Activities*, *Services* and *Intents*, allowing for transparent interactions of application code in both local and distributed settings. Quality-of-Service requirements of both applications and reconfiguration services are handled by supporting an underlying operating system QoS Manager module.

This framework will be used to develop the real-time capabilities of the Android operating system and particularly to develop adequate strategies for multiple parameter quality-of-service of applications (considering both tasks and communication streams). We also plan to investigate further on how to better manage dynamic adaptations during reconfiguration/mobility phases. An implementation of the framework proposed in this paper is available at [5].

Acknowledgment. This work is partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência e a Tecnologia - FCT) and project RESCUE (PTDC/EIA/65862/2006).

References

1. Preuveneers, D. and Berbers, Y. (2010) “Context-driven migration and diffusion of pervasive services on the OSGi framework”, International Journal of Autonomous and Adaptive Communications Systems, Vol. 3, No. 1, 2010, pp. 33-22.
2. Malek, S., Edwards, G., Brun, Y., Tajalli, H., Garcia, J., Krka, I., Medvidovic, N., Mikic-Rakic, M., Sukhatme, G., ”An Architecture-Driven Software Mobility Framework”, Journal of Systems and Software, special issue on Software Architecture and Mobility, Vol. 83, Issue 6, Jun. 2010, pp. 972-989.
3. Nogueira, L., Pinho, L., "Time-bounded Distributed QoS-Aware Service Configuration in Heterogeneous Cooperative Environments", Journal of Parallel and Distributed Computing, Vol. 69, Issue 6, 2009, pp. 491-507.
4. Maia, C., Nogueira, L., Pinho, L., “Evaluating Android OS for Embedded Real-Time Systems”, to be published in the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2010), 2010.
5. Distributed and Mobile Framework for Real-Time Systems (DiseRTS), http://www.hurray.isep.ipp.pt/activities/RTSoft/distFrameworkOverview.ashx/, July 2010.

Dependable Perception in Wireless Sensor Networks

Luís Marques[1] and António Casimiro[2]

[1] FC/UL* lmarques@lasige.di.fc.ul.pt
[2] FC/UL casim@di.fc.ul.pt

Abstract. In this paper we present an analysis of perception dependability in wireless sensor networks. We put forward a model of perception and present the necessary definitions to understand and ascertain perception quality and dependability. We also present some directions for future work aiming at providing middleware support for perception dependability, using this perception model.

Abstract. Neste artigo apresentamos uma análise sobre confiabilidade na percepção em redes de sensores sem fios. Descrevemos um modelo de percepção e apresentamos as necessárias definições relativas à qualidade e confiabilidade da percepção. Indicamos também direcções de trabalho futuro com o objectivo de fornecer um middleware de supporte à percepção confiável, usando este modelo de percepção.

1 Introduction

Much work has been done in the last decade regarding distributed systems and applications based on networked sensors and actuators. Specifically, there has been great interest in wireless sensor networks (WSNs), in particular focusing on environment monitoring [1] and target tracking applications [2], through the use of small, low powered radio-enabled sensor nodes.

The general research area continues to garner attention, as can be seen through the current use of fashionable terms such as "cyber-physical systems" and "Internet of things", which refer to systems where information technology pervades and closely interacts with the real world environment. In fact, the real-time nature of information processed in these systems creates difficult problems, in particular if WSNs are to be used in sensor network-based control applications. Our present work is directed towards addressing some of these problems.

Despite the existence of a large body of work in this area, research has been limited to some specific topics, such as platforms and OS support, spacial coverage and awareness, synchronization, security, communication protocols and energy conservation (see [3] for a good survey). Overlooked in this research has been the issue of data quality, being either taken for granted or addressed with ad hoc validity mechanisms, mostly by using synchronized clocks [4] and setting validity deadlines for data, or simply by overwriting old data [5].

Lacking has been a systematic analysis of faults that may impair perception reliability and the definition of abstractions to allow reasoning in terms of perception quality. More practically, it is necessary to develop algorithms and mechanisms to deal with faults and improve perception quality and dependability. This can be done as part of a programming

* This work was partially supported by FCT through the Multiannual Funding and the CMU-Portugal Programs.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 715–718

model and middleware framework which would allow systems to be designed to explicitly reason about and be aware of perception quality and data validity.

In this paper we present a perception model upon which a middleware for dependable perception in WSNs can be built. We also present some possible directions for implementing such middleware.

2 Perception Model: Definitions and Dependability Issues

In our perception model we distinguish between environment entities and the systems' perception of them. We adopt the terminology in [6], referring to the former as *Real-Time entities* (RTe), which conceptually represent the actual state of some environment variables, exactly as they exist in a given moment, and to the latter as *Real-Time representatives* (RTr), which are the internal representation of a RTe, as imperfectly perceived by the system.

Having this RTe/RTr duality, we can thus define *perception quality* (or *observation error*) as the similitude (or difference) between a RTr and its respective RTe at some given time instant. The more similar a representative is to its RTe, the higher the perception quality.

Many applications require a minimum level of quality, so it is important to be able to distinguish between scenarios where an RTr is sufficiently similar to the corresponding RTe and scenarios where it is not. We can thus refer to *perception validity*, or *temporal consistency* of an RTr with respect to the RTe, which indicates if the perception quality is sufficient for an application. More formally, given a maximum allowed error ϵ, the perception is valid, or the RTr is temporally consistent at t, if $|\mathrm{RT}_r(t) - \mathrm{RT}_e(t)| \leq \epsilon$.

This definition of validity requires possessing knowledge of the real value of the RTe, and as such is suitable for theoretical reasoning, but cannot be directly used in practice (the state of the RTe is obviously unknown to the system). Instead, one or more of several different approximations can be used to derive the validity of an RTr. A suitable approach is to set a temporal validity interval or a deadline as soon as the RTe is observed (or sampled), after which the RTr value can no longer be considered valid. This approach requires a global notion of time in the distributed system and is dependant on the RTe's dynamics. Therefore, while it is always possible to perceive the state of any environment, doing so in a dependable way requires its dynamics to be somehow predictable, or no guarantee can be given about the validity of its perceived state. We thus consider as part of our perception model that we can bound the environment dynamics.

We consider that the perception of real-time entities can be affected in two ways. One is by faults and measurement errors affecting the sensing processes, such as malfunctioning transceivers and intrinsic sensor tolerance intervals. The other is by faults and temporal uncertainties affecting the communication in the wireless sensor network.

In this model we consider that faults are stochastic. We can then determine the best possible perception quality that is achievable with some probability, by considering the best-case sensing outcomes and the best-case transmission delays between information producers (sensing RTes) and consumers (using RTrs).

However, we need to distinguish the following two cases: either the system is able to eventually provide an RTr within the required error margin or it will never be able to do so. This second case would happen in situations where the fidelity of the sensors is insufficient for the required error margin or the transmission times are too large, always preventing updates of an RTr to be done in useful time.

If a system is able to eventually provide a valid RTr then the issue is that this validity cannot always be maintained, only being secured during some periods. In other words, it is

possible to reason in terms of the probability that an RTr is valid at a given instant. We call this the *coverage* of an assumed error margin, which is a measure of the quality of service (QoS) provided by the WSN. In fact, we can generalize this concept so that completely unworkable QoS requirements are simply classified as being provided with zero coverage.

Redundancy can be used to deal with faults in WSNs and increase dependability. This can be spacial redundancy (the same information transmitted through different paths) or temporal redundancy (waiting more time to receive new updates, despite some lost ones). The impact of these approaches on the perception quality has to be investigated. For instance, waiting times in intermediate nodes during message propagation may be useful to overcome lost messages, but also imply a perception quality degradation.

Given the described model, middleware support can be provided to let applications enjoy a requested level of perception quality (ensure temporal consistency with a certain probability), or become aware of the maximum achievable quality. Providing awareness and probabilistic guarantees is a way to improve system dependability.

3 Middleware for Dependable Perception

Using the previously described perception model, a middleware for dependable perception can be built, which allows WSN-based applications to state their requirements concerning perception quality and coverage and have local real-time representatives be automatically updated accordingly. In the following paragraphs we sketch some general and preliminary ideas about what may be done by such distributed middleware.

The application specifies its requirements at a sink node. Then, before any actual sensor data dissemination occurs, the middleware undergoes a setup phase to propagate the requirements down the network to the relevant entities, i.e., active sensor nodes (which observe the RTe) and routing sensor nodes (which only propagate sensor data). A specification of the RTe dynamics must also be provided by the application at the sink node. When the network is configured, then the middleware enters a steady-state phase, during which sensor data is disseminated using a protocol that is designed to achieve the required quality/coverage specification of the RTr at the sink node. A key issue will be to manage the trade-off between the involved dissemination costs and the required dependability.

Depending on the nature of the considered RTe, there can be one or several active sensor nodes, performing actual sensing of the RTe (converting a physical state into its digital description). Every sensor node may also act as a routing node, propagating RTr updates to sink nodes. However, these different roles can be abstracted at the middleware level. Incoming data should be treated homogeneously, whether it has been received from a peer node, or has been generated locally by the sensing and transducing hardware. The task of the middleware is to decide when to propagate the data further and to which nodes. The decision will depend on several issues, including the dependability requirements and knowledge about the state of the network, failures, the number of hops to reach the sink and the actual validity of the received data. In essence, all the variables with impact on the validity of data should to be taken into account. In particular, this means that the middleware will also incorporate mechanisms to evaluate the operational context, which can have a simple local scope, or may involve distributed operations as well.

A possible functioning of the dependable perception middleware is then as follows. During the setup phase it is necessary to evaluate if the application request is feasible. By monitoring network conditions and using a fault probability model, the nodes are able to compute

the amount of time that is necessary to ensure, with a given coverage, that RTr updates propagate between sensing nodes and sink nodes. If the middleware determines that a request is feasible then the sensor data dissemination mechanism can start. After the setup phase we can rely, with the specified coverage, on the RTr validity in the sink node. Preserving this coverage during the steady-state phase implies ensuring that each time the sink node's RTr is about to become invalid it should receive a RTr update which extends its validity. Intuitively, and without further concerns, sensor data generation and propagation should be as frequent and fast as possible, thus always ensuring the best possible RTr validity. However, this could be too resource consuming (in WSNs the objective is always to save resources) and even prejudicial to the objective of securing a given data validity, by creating network contention. Therefore, the middleware must incorporate mechanisms to evaluate the precise amount of resources that are needed to secure the application requirements, forwarding sensor data only when necessary and only through the strictly necessary redundant paths.

Given the dynamic and uncertain nature of the operational conditions, the potential need to accommodate topology changes and the heterogeneity of context perceived by each node, it is important that the middleware at each node continuously evaluates the amount of resources needed for data propagation. For instance, if no faults occur and an update is quickly propagated to an intermediate sensor, that sensor might recognize the slack still available for transmission and conclude that it can temporarily delay the propagation of the update. It could thus save energy if a subsequent update of the same RTr arrived at the node during that delay, thus improving the overall trade-off between validity and cost.

4 Conclusion

How to be aware and how to assure perception quality is still an open problem, relevant to many types of WSN applications. This paper discusses this problem and tries to provide some preliminary ideas on how to deal with it at the middleware level.

References

1. Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D., Hong, W.: A macroscope in the redwoods. In: SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems, ACM Press (2005) 51–63
2. Simon, G., Maróti, M., Lédeczi, A., Balogh, G., Kusy, B., Nádas, A., Pap, G., Sallai, J., Frampton, K.: Sensor network-based countersniper system, ACM Press (2004) 1–12
3. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Comput. Netw. **52**(12) (2008) 2292–2330
4. Rhee, I.K., Lee, J., Kim, J., Serpedin, E., Wu, Y.C.: Clock synchronization in wireless sensor networks: An overview. Sensors (2009) 56–85
5. Williamson, G., Cellai, D., Dobson, S., Nixon, P.: Modelling periodic data dissemination in wireless sensor networks. In: EMS '09: Proceedings of the 2009 Third UKSim European Symposium on Computer Modeling and Simulation, Washington, DC, USA, IEEE Computer Society (2009) 499–504
6. Veríssimo, P., Rodrigues, L.: Distributed Systems for System Architects. Kluwer Academic Publishers (2001)

Sistemas Inteligentes

Decision Making for Agent Moral Conducts

Helder Coelho [1], António Carlos da Rocha Costa [2] and Paulo Trigo [3]

[1] LabMAg, Faculdade de Ciências da Universidade de Lisboa, 1749-016 Lisboa, Portugal
hcoelho@di.fc.ul.pt

[2] Centro de Ciências Computacionais, PPGMC, Universidade Federal do Rio Grande, 96.201-900 Rio Grande, RS, Brazil, ac.rocha.costa@gmail.com

[3] LabMAg, Instituto Superior de Eng. de Lisboa, DEETC, 1959-007 Lisboa, Portugal,
ptrigo@deetc.isel.ipl.pt

Abstract. Looking to the operation of an agent architecture, ie. its goal generation and maintenance processing, is relevant to understand fully how a moral based agent takes appropriate and diverse decisions within social situations of serious games. How decision does happen is a complex issue and the major motivation of this paper, and our answer, the proposal of a new architecture, is supported on the clarification of the organization and structure of an agent, ie. the interpretation of agent actions (moral-driven behaviour) under the pressure of severe constraints.

Keywords: moral architecture, values and norms, behaviour regulation, morality reconsideration.

1 Introduction

"Synthesis and simplification are the essential issues in architecture".
Alejandro Aravena, Revista Única Sep. 19, 2009.

Recently, we have been discussing the proposal of an overall architecture of moral based agents, embedded in a social multitude, by facing two major issues, intelligence (Corrêa and Coelho, 2010) and complexity (Coelho and Costa, 2009; Coelho et al, 2010). This line of research is different from the logical programming direction, and it is more akeen to Sloman and Minsky bet on an architecture for cognitive diversity. The follow-up of a recent R&D EEC project, EMIL (2007-09), help a lot to clarify differences between norm-governed and moral agents. Norm processing is almost trivial as compared with moral decision. Several conjectures were put forward:

C1: A moral agent, like any cognitive agent, is defined by an hybrid architecture.
C2: The key, and central question, concerns how its decision policy is managed, because the environment is pro-active and it requests a certain complexity of social behaviour.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 721–732

C3: The architecture has many layers, at least four systems (cognitive, emotional, moral, and esthetical), and there are many interactions (feed-forward and feed-back flows) among its modules before an appropriate (moral) decision is attained.
C4: Choice and preference reconsideration (action selection) is mandatory. Moral agents have different individual cultures and values and must be cautious and respectful in order to avoid inappropriate behaviours (generation of social conflicts).
C5: Behaviours are ruled by norms which depend on values. Norms are means in order behaviours be compatible to moral values.
C6: Moral global behaviour is the result of many informed local decisions, taken by different modules and along n-layers, of feed-back and feed-forward moves, and the negotiation among those modules is often required to support the final decision.

2 Morality

A vision of morality, conformity to the rules of right (moral, virtuous) conduct, the so-called moral character (evaluation of particular individual qualities) is strongly connected to what is socially defined as normal or appropriate (March and Olson, 2009), true, right or good, in spite of the necessary calculus of consequences and expected utility. How agent conduct is engendered, according to values, rules, codes and principles is only a fraction of what is necessary. Any agent acts because it pursues to achieve a purpose or satisfy a desire, and it seeks also adequate actions in defence of its interests and, often, anticipates future consequences following criteria of similarity and congruence, rather than likelihood and value. Appropriateness reflects learning of some sort from personal history, but it does not guarantee technical efficiency or moral acceptability.

When moral judgments (weighing reasons for and against affective attitudes and moral intuitions) are given, in face of some non-trivial situation (eg. switch dilemmas in trolley problems), an intriguing contrast emerges between the intuitive opinions from those considering the scenarios. The respective acts may be evaluated differently and the choices ("sacrifice one life in order to save five") are unexpected for similar events. The interpretation can be explained by emotional arousal and by the importance attributed to intuitions. So, the utilitarian or deontological views are in danger to be good candidates for supporting moral judgment. We are convinced that somewhere in between lies the sound solution.

Are moral issues just a matter of taste or culture? Are moral judgments provoked by expressions of affective states on which reasons have little influence? Or, should more attitudes be justified, ie. some moral and cultural judgments are wrong, and others right, because they relate to facts of moral relevance in adequate or inadequate ways. What ought we do? Ignore all our ordinary moral judgments, and do what will produce the best consequences, or follow what we were told to do!

Morality is the respect for the other, and it is not a monolithic concept with sharp boundaries. Really, what happens in moral judgment, is mostly a part of typical response patterns (the so-called moral signature, full character of an agent), because there are aspects of acts and/or situations that are relevant to take moral decisions.

3 Moral character

When studying moral agents we are attracted by a diversity of feelings, a kind of pro-social sentiments, of guilt, compassion, empathy, anguish or ambivalence, triggered by states of affiliation or sadness. Moral decisions may be very complex because they entail the cooperative interplay of several systems, namely of thought, emotion, empathy or foresight, and along layers of importance. How can we design such an agent able to make judgements, by juggling evidence and emotions, reasons and sentiments? How may we envisage a moral mind? Three directions are possible: 1) With a set of mathematical formulae used to make predictions about behaviour? 2) With a computer program to simulate thinking? or, 3) With a description (operation) of mechanisms that explain observed mental phenomena? Our conjecture is: with a decision apparatus, and following the third trend.

Morality is more than simple utilitarianism or deontology, as some authors defend (Hauser, 2006), focusing on what actions are morally, right or wrong. It is not only a utility function with some devious calculus of importance, because it requests emotional regulation, according to recent findings of Cognitive Neuroscience, and the full cooperation between reason and emotion, at least.

We judge our actions by imagining what the future looks like, and we act because we would like to achieve a purpose, preserving a set of qualities. Imagination is essential to empathy, in order to comprehend the full moral dimension of a situation, and, in point of fact, to be an agent with moral virtues of character it is necessary to have more than general principles of rationality. And, get a direct answer about how the mind works implies to get closer about the description of several mechanisms that explain the mental phenomena at large (Minsky, 2006).

The most successful theoretical explanation in cognitive science has been mechanistic in the sense elucidated by philosophers of science. A mechanism is a system of parts whose interactions produce regular changes. Therefore, the idea of composing the architecture of a moral agent (our proposal in this paper) is debatable, but it allows to design how an agent achieve a variety of conflicting aims (components), such as: deliberation, advance goals and act on commitments that must be revisable; action guided by context-sensitive judgement; ability to be sensitive to the requirements of particular circumstances; emotional connection, or sensitivity of moral concepts (moral attention), imagination and self-reflection (Singh and Minsky, 2005). Such an architecture requires all of the skills we associate with general intelligence and common sense reasoning, namely 1) reasoning ability, ie. making logical inferences, synthesizing and interpreting information, or recognizing similarities and differences; 2) getting vision of the situations, judging and doing accurate predictions; 3) moral perception with intuitive skills of situations embedded in social customs, personal and relation histories (social interactions); 4) correcting and revising power (truth maintenance capacity) in order to guide further/future judgements (Ethics versus centred on wanting something other than what exists); and, 5) emotional intelligence in order to preserve the value of options not acted and to guide the agent through the practical reasoning process.

4 Moral totality

Usually, a moral decision does not follow a single criterion. It requires comparison of different points of view, some in favour and some against, and an algebra to take care of multiple criteria and trade-offs. So, amalgamating the multidimensional aspects of the decision situation into a single scale of measure is no longer the way out. Prudential calculus is made by characters served by a set of qualities, which implies taking into consideration some personalities of an agent to cover the skills akin to morality (personal, cultural, affective, anticipatory).

Decision is often defined as an objective function like a single point of view (profit or cost index) representing the preference (or not) of the considered actions, which is maximized (or minimized). In moral contexts, this is very simplified and unnatural, because any decision is always related to a plurality of points of view, and the pros and cons (relevance) are to be taken in due account. So, it is advisable to make the aggregation of individual preferences into collective ones when choosing, ranking or sorting the actions (solutions, alternative courses…).

Our intuition, about the good choice, is on multiple criteria decision analysis (MCDA) because there are several dimensions involved apart of ethics, it is suitable to structure the complex evaluation and to include both qualitative and quantitative criteria. This choice favours a behaviour that will increase the consistency between the evolution of the process, the objectives and the values. By attending each pertinent point of view separately, independently from the others, it is generally possible to arrive at a clear and common elicitation of preferences regarding the single point considered. This also leads to associating a specific criterion to each point of view.

5 Around the design of a moral agent

The interplay of mentality (cognition), sociality (collective regulation) and morality (norm/value guidance) reveals the definite anatomy of those smart creatures able to think about, to interact with others in a society, and also to decide upon good and evil. Which is the most suitable architecture for an agent with these three features?

Agents can be reduced to simple bit strings when genetic programming is adopted in social simulation of complex scenarios. In what concerns symbolic programming, an agent can be more elaborated than a decision (utility) function. For example, in a risky environment, (Castelfranchi et al, 2006) adopted a two-layers structure by mixing an extended BDI with an emotion manager for modelling cautious agents. In order to design social and normative agents (Castelfranchi et al, 2000) defended norms as meta-goals on the agent´s own processes, around a BDI kernel and two levels of process abstraction. In the serious game (mixing human and artificial agents) of water management (Adamatti et al, 2009), any artificial agent had a behavioural profile linked to one or more strategies regarding a certain role (the BDI model was simplified), no learning and planning modules were available, and only reduced decision making skills were offered, and again a one-layer structure was adopted. In other serious games on participatory management of protected areas (Briot et al, 2008), conflict dynamics was taken care and a more advanced decision capability was

implemented, but agents had no mentality and affective power. When designing cultural agents, (Mascarenhas, 2009) updated an old architecture of social intelligent agents for educational games and proposed to combine a memory store with a reactive device and a deliberative machine, without forgetting the motivational states of the other agents. However, serious games require moral agents, to be acceptable by users.

A moral agent, as it was defended by Hauser and by Green, is a mix of cognitive and affective capabilities, but no architecture was till today presented as the definite one, despite several design attempts (Wiegel, 2006; Andrighetto et al, 2007), without any explanation on the operation of the moral decision machinery. Several questions needed yet to be answered: What makes a moral (norm-abiding, virtuous, conventional) agent? By what mechanisms and layers can abstract moral principles and values spread or decay from one agent to another (like memes)? How are explicit morals implemented and added to the overall architecture to generate aims and desires, and, later on, to fix conducts? What is the function of moral reasoning, of perceiving a new detail in a situation, or of understanding the moral relevance of what we see? Which is the specific role of the (cognitive, moral, ethical) values?

A moral agent needs to get a more intricate way of thinking than a simple reactive (assimilate observations of changes in the environment) or a proactive one (reduce goals to sub-goals and candidate actions). Why? It is not sufficient to embody a goal-based or a value-based model. We need a mix of intuitive (low level) and deliberative (high level) processes, and also the ability to think before acting (pre-active) when choosing between right or wrong, ie. capability to think about the consequences of the candidate actions (generate logical consequences of candidate actions, helping to decide with heuristics or decision theory between the alternatives). The classic component based on the observe-think-decide-act cycle (present in the BDI model) is unable to deal with morality because we get different kinds of goals (achievement, maintenance) and, at the same time, preferences and priorities are requested.

The one-layer structure is no longer the correct solution because we arrive at our ultimate moral (utilitarian, where results maximize the greatest goods, or deontological, where any moral evaluation is independent of consequences) judgements by a mix of emotions and conscious reasoning. As a matter of fact, emotions drive behaviours as weights, and play a critical mediating role in the relationship between an action's moral status and its intentional status. A moral ability may be seen as a set of rules (a grammar according to Hauser) to constrain the behaviour of the agent: each rule having two ingredients, the body of knowledge and the set of anchored emotions, which are going to interplay. See our proposal for the architecture of a moral agent in figure 1.

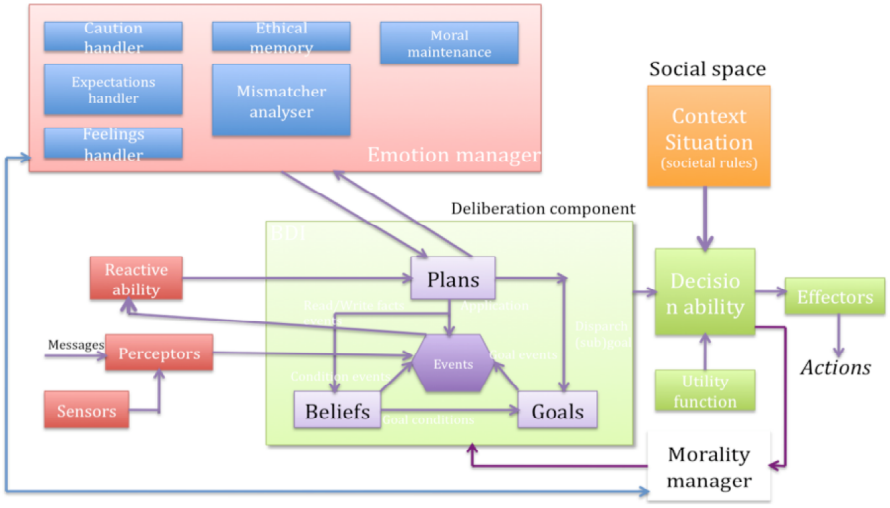


Fig. 1. Proposal of a moral agent kernel architecture.

This tentative proposal of a highly modular and hybrid moral architecture is composed by three layers, as opposed to two of the deliberative normative architecture of (Castelfranchi et al, 2000): 1) the first, for the classical cognitive flow, based upon deliberation (BDI), 2) the second, for the moral system with judgement (upon choices) and decision, a moral maintenance system, an ethical memory, and the morality (including a moral grammar and a moral learning module) manager, and 3) the third one for the emotional system containing the emotion manager (including three handlers for caution, expectations, and feelings, and a mis-matcher analyser). The two managers interact heavily between them and, also, each one with the BDI and decision modules.

The architecture of figure 1 has a high-level (the moral reasoning), mainly concerned with how the agent manages its currently available best options for diverse social situations, ie. how it orchestrates the choices together into a moral coherent behaviour. Such a structure allows the moral agent to be flexible enough in changing social environments and to adapt graciously. And, a low-level (moral reaction): a moral judgement is the consequence of a rational process (based upon moral rules) applied to a certain situation or of a simpler reactive process. The moral agent's decisions are not rigid ones but rather well balanced decisions, weighing preferred options or choices, with the aid of a morality manager (the white box in figure 1). The involved mixture of intuitive and deliberative processes embody also a question of power: who is in charge of the higher or lower levels?

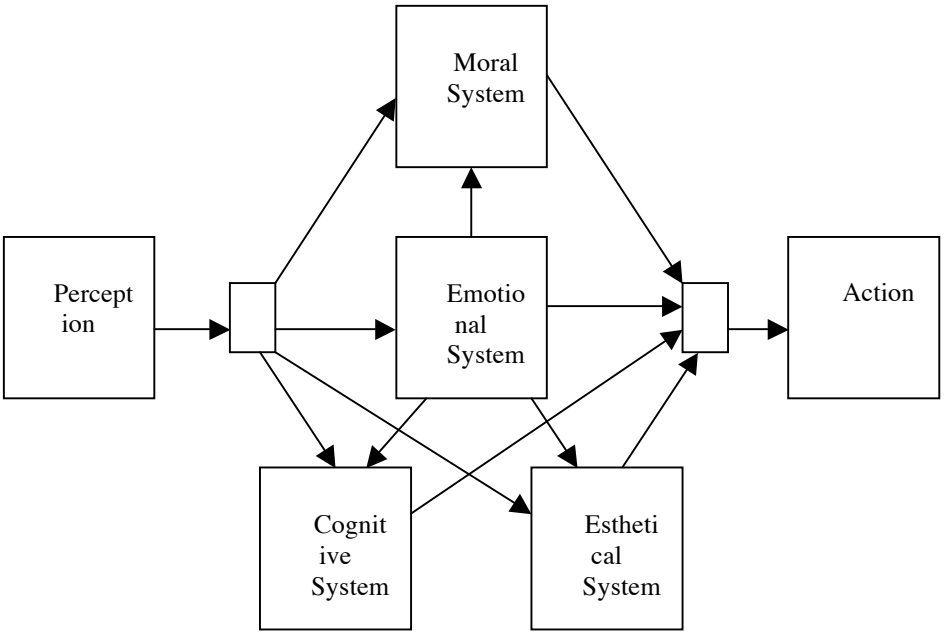


Fig. 2. Interplay of diverse systems

According to the social intuitionist model by (Green and Haidt, 2002), there is more one layer (Esthetics), because moral judgment is similar to esthetical one: when we listen to a story or look to some action/behaviour we get an instantaneous feeling (intuitions with some affective value) of approval/disapproval. In figure 2, we sketch an extended moral architecture with four layers, where the esthetical layer is appropriate to support the agent social reasoning, which involves, as a matter of fact all the four layers (each one with its own objectives and values)! Such an extension aims to model adequately the phenomena behind moral decision working, including the processing of all causal mechanisms.

We adopted an idea from (Dignum et al, 2001) by using desires (self purposes) as links between the cognitive layer and the other ones. Desires are generated by the non cognitive layers and work as factors (to be mixed with the normative factors) and capable to influence the agent deliberation. In (Castelfranchi et al, 2000), norms were mental representations (objects) entering the mental processing and the interaction, in several ways, with beliefs, goals and plans in order to fix the agent´s behaviour. This is also an interesting idea, adopted in the operation of our 4-layer architecture to allow an agent may follow or violate norms. In (Corrêa and Coelho, 1998) we proposed a table of mental states of an agent, facilitating the inclusion of other mental objects (eg. expectations, hopes), and extending easily the BDI classical architecture (Cascalho, 2007) with mental states through attributes (a kind of weights), laws of composition and control mechanisms).

Those desires are relevant to constrain moral judgments. By adopting influence diagrams, we may connect them to judgments by arcs, where each one has a weight

according to the rules associated with the agent objective, emotional or cognitive situations. A moral judgment can be positive or negative, depending on deductions made by the moral rules, having a certain intensity/importance given by the sum of the weights of those factors (very high, high, average, low, very low).

Every decision an agent makes, when it comes to choosing between right or wrong, reveals his true character (subjectivity and identity): the Humean model with emotions behind judgements or the Rawlsian model with emotions and reasons after judgements have only two layers, where the main processing flow is done sequentially in one-layer, and the trade-offs are not allowed. In a 4-layer architecture, the interactions among layers, systems and components (eg. emotional vs. moral systems) make the personality of an agent. There is always a sentiment of avoidance in violating what seems to be reasonable, ie. the possibility to have access to the outcomes (classifications) of the agent actions.

An effective decision should be based on the achievement of objectives. Criteria (universal principles, values, beliefs) and objectives (purposes, aims, desires) are used to measure how well we achieve our goals. Decision making is always difficult because trade-offs must be made among competing objectives. In order to consider trade-offs, we must be able to evaluate and measure each aspect of the decision, some quantitative, some qualitative, some very important and some not so important. Uncertainties and competing interests among the components (deliberation, emotion, morality, decision) also add to the complexity of the overall decision making.

A moral agent associates always reason with emotion, social values and cultural-situational knowledge before making a decision. Therefore, its more-than-one-layer architecture, integrating micro and macro levels, requires an extended (with will and expectations) BDI model, the addition of emotional machinery to deal with sentiments, a library of contexts to situate any evaluation, heuristics to avoid wrong decisions (mind traps), a sort of universal moral grammar to fix any sort of moral system and action generation, and also modules concerning decision taking, constraint satisfaction (reinforcement) learning and planning. The organization with interconnected multiple layers seems inevitable on account of the balance between reasoning and emotion and the assembling/tuning of composite judgements (embedded in preference criteria).

6 An illustrative experiment

The interplay of cognition, collective regulation and norm/value guidance is better described by an example that justifies the components of our proposal. The usual purpose of a fairy tale (fable) is to provide a context for some general moral interpretation. Although the global message is usually very clear, a deeper reading of some fable details often reveals ambiguity even at the morality level interpretation.

We consider the well-known “Jack and the Beanstalk” fable (1807, British unknown author). The story tells of Jack, a very poor boy, whose lack of common sense exasperates his widowed mother. She sends him to the market to sell their only possession, a cow, but along the way, Jack meets a stranger (adult) who offers to buy the cow for five "magic beans". Jack is thrilled at the prospect of having magic beans,

so he makes the deal. When he arrives at home with his beans, his mother, in despair that they were ruined, throws the beans out of the window, and the two go to sleep without eating any supper. The story proceeds with several adventures but, in the end, the boy and his mother get very wealthy because the beans turned out to be really magic.

The story fragment of the "cow for beans' trade" illustrates some interactions between goals, plans, beliefs, desires, social norms and moral values. We named the two agents J and B, respectively, referring to Jack (a child) and the adult owner of the (magic) beans, so we have Ag = { J, B }. The set of available resources may be described by Rs = { cow, beans, food, money }. The "possess" relation, p: Ag → Rs, describes an agent's belongings, thus p (J) = { cow }, and p(B) = { beans, food }. Each agent's goal is described by g: Ag → Rs, therefore g (J) = { money, food } and g(B) = { money }. According to the story, a general plan for each agent may be devised as follows: plan(J) = [get(cow), exchange-for(cow, money), buy(food)] and plan(B) = [get(beans), exchange-for(beans, $money_1$)].

Additionally, a social norm underlying the whole story is that "an adult always *negotiates honestly* with a child". This norm holds two important concepts: a) the negotiation, and b) the honesty. The "negotiation" calls for utility based reasoning and the "honesty" resorts to the moral interpretation of one's motivations. We know that the utility for a cow is much higher than the utility for five beans, i.e., util(cow) >> util(beans). But, how does the "honesty" concept integrates the overall formulation? One alternative is to interpret "honesty" as a moral evaluation of some subset of the agent beliefs, i.e., moralEval: 2^{Bel} → [0,1], where Bel represents the belief set and 0 (zero) and 1 (one) represent, respectively, the least and the most adherence to the moral principles underlying the corresponding belief subset. Additionally, the "moral signature" of each agent relates each moral concept, e.g., "honesty", with a subset of beliefs, i.e., moralSig: Mc → 2^{Bel} where Mc is the set of moral concepts (within a certain domain). Cleary, this moral signature relation, moralSig, already expresses some of the "moral guides" behind the (human) designer of the relation. Thus, a complex domain requires a (human) designer sensibility and expertise to be tuned in the process of interacting with other (human) designers.

Let us describe agent J as follows: "util(cow) >> util(beans)" ∈ Bel_J, the agent moral evaluation is $moralEval_J$({ util(cow) >> util(beans) }) = 0 and its moral signature is $moralSig_J$(honesty) = { util(cow) >> util(beans) }. So, agent B, after meeting agent J, refines its original plan into a new plan'(B) = [get(beans), exchange-for(beans, cow), exchange-for(cow, $money_2$)]. From a purely utilitarian perspective, $money_2$ must be higher than $money_1$ (above a threshold) in order for agent B to pursue this new more complex plan. But "util(cow) >> util(beans)" ∈ Bel_B so agent B is willing to drop the original plan and adopt the new one (plan'). But, at the same time, the negotiation environmental context includes a child so the "*negotiate honestly*" norm (cf. above) becomes active. Now the agent must apply its moral signature, moralSig, regarding "honesty". The agent uses a machinery to combine the moral evaluation, moralEval, with additional parameters, such as the utilitarian added-value for the new plan. Here, we simplify and decide just upon the moralEval; in this scenario we have $moralEval_B$({ util(cow) >> util(beans) }) = 0 so the agent B proceeds and moves to the new plan.

Now, under plan'(B) agent B must find a way to convince agent J that trading a cow for beans is a fair trade. Therefore, B must raise J's expectations regarding the beans, and B believes that a way to raise a child's expectations is to invoke directly its "magic world". Hence a new plan is generated plan''(B) = [get(beans), inform(beans, "magic"), exchange-for(beans, cow), exchange-for(cow, $money_2$)]. The new plan takes B to convince J to trade its cow for the beans; B gets the cow's money and J makes a plan to get a lot of food and richness from the beans.

This illustrative scenario shows agent B moving forth and back among plans, beliefs, social norms and moral signatures and values. But, if one is not able to follow all the internal reasoning details is it possible to know whether agent B was following norms and moral principles? Let us assume that agent B believes that the beans are really magic and that they would provide a huge fortune to its owner. Then a completely different scenario would arrive and the only difference (from the previous one) could be that "util(cow) << util(beans)" $\in Bel_B$. In this new scenario agent B exhibits an aesthetically altruistic behaviour. In fact such a high order altruism also resorts for some degree of divine power over disgrace and poorness. But, on the other hand if we were to dissect the child's beliefs and (utilitarian and moral) reasoning we could find that a social norm such as: "trading extremely differently valued assets is not a fair trade" is also active. Usually this is a norm that a child knows about in order to prevent him from bargaining with a much younger child. In this new context the child also ends up bypassing its "fairness" moral signature along with the associated social norm.

The above reasoning scenarios were drawn from a deeper analysis of the internal processes of two agents in the context of an apparently innocuous fairy tale.

7 Conclusions

> "Agents are a way of thinking, a conceptual frame for modelling active, distributed, complex, and layered phenomena."
> C. Castelfranchi in IEEE Internet Computing, March-April 2010.

The research and experimentation around the sketch of an architecture for moral agents is supported on the belief that moral decisions are very complex processes. Applications such as regulation of e-communities or realistic serious games for managing human capital are eager of new agent models and architectures with ethical concerns and some sort of subjectivity. We invested, for more than a decade, in heavy experimentation about agent models and architectures, for individual and collective decision making (large scale disasters, electric energy markets, semantic web spaces), trying in each step forward, to increase the number of interactions and relations among components of the next architecture.

The character of a moral agent is dependent on its architecture, namely on the interactions (for negotiation) and on the relations (global complexity) among its components. Any architecture reveals also a mix of high level (deliberative, moral reasoning) and low level (reactive, intuitive) processes, where some one of them is in power to support the acting.

Our agent design ideas were based on the understanding of the semantic operation of morality, rethinking computing and knowledge in terms of interaction and social processing, but several open questions frame still our current research: How can we operationally verify all the interactions behind a moral agent architecture? How do actors produce and are at the same time a product of social reality? Which ideas are accepted and which are rejected driven by adaptation and evolution? How many are slowly assembled from diverse data in a single mind? Answers, from Cognitive Neurosciences, Moral or Evolutionary Psychology, point to a strong focus on a context sensitive approach to agency and structure, the interplay of which leads to emergent phenomena, underlining the generative paradigm of computational social science. Agent-based modelling and simulation can be of great help in order to allow a better comprehension of this sort of complexity.

References

Adamatti, D., Sichman, J. and Coelho, H. An Analysis of the Insertion of Virtual Players in GMABS Methodology Using the Vip-JogoMan Prototype, Journal of Artificial Societies and Social Simulation, JASSS in press, 2009.

Andrighetto, G., Campenni, M., Conte, R. and Paolucci, M. On the Immergence of Norms: a Normative Agent Architecture, Proceedings of AAAI Symposium, Social and Organizational Aspects of Artificial Intelligence, Washington DC, 2007.

Briot, J.-P., Vasconcelos, E., Adamatti, D., Sebba, V., Irving, M., Barbosa, S., Furtado, V. and Lucena, C. A. Computer-Based Support for Participatory Management of Protected Areas: The SimParc Project, Proceedings of XXVIIIth Congress of Computation Brazilian Society (CSBC´08), Belém, Brazil, July 2008.

Cascalho, J. The Role of Attributes for Mental States Architectures, PhD Thesis (in Portuguese), University of Açores, 2007.

Castelfranchi, C., Dignum, F, Jonker, C. M. and Treur, J. Deliberative Normative Agents: Principles and Architectures, Proceedings of 6th ATAL Conference (1999), Intelligent Agents VI, Springer LNCS 1757, 2000.

Castelfranchi, C., Falcone, R. and Piunti, M. Agents with Anticipatory Behaviours: To Be Cautious in a Risky Environment, ECAI, 2006.

Coelho, H. and Costa, A. R. On the Intelligence of Moral Agency, Proceedings of the Encontro Português de Inteligência Artificial (EPIA-2009), October 12-15 Aveiro (Portugal), in L. S. Lopes, N. Lau, P. Mariano e L. M. Rocha (eds.), New Trends in Artificial Intelligence, pp. 439-450, 2009.

Coelho, H., Costa, A. R. and Trigo, P. On the Complexity of Moral Decision, FCUL and DI Working Report, 2010.

Corrêa, M. and Coelho, H. From Mental States and Architectures to Agents´ Programming, Proc. Of the 7th Iberoamerican Congress on Artificial Intelligence

(IBERAMIA98), Lisbon 6-9, Springer-Verlag LNAI 1484, pp. 64-85, 1998.

Corrêa, M. and Coelho, H. Abstract Mental Descriptions for Agent Design, Intelligent Decision Technologies (IDT), an International Journal, IOS Press, 2010.

Costa, A. R. and Dimuro, G. Moral Values and the Structural Loop (Revisiting Piaget´s Model of Normative Agents), PUC Pelotas Working Report, 2009.

Dignum, F. Kinny, D. and Sonenberg, L. From Desires, Obligations and Norms to Goals, Utrecht University, 2001.

Green, J. and Haidt, J. How (and Where) does Moral Judgment Work? In Trends in Cognitive Sciences, Academic Press Volume 6, Issue 12, December 2002.

Hauser, M. D. Moral Minds: How Nature Designed our Sense of Right and Wrong, Ecco/Harper Collins, 2006.

March, J. G. and Olsen, J. P. The Logic of Appropriateness, Arena Centre for European Studies Working Papers WP 04/09, University of Oslo, 2009.

Mascarenhas, S. F. Creating Social and Cultural Agents, IST MS.C. Thesis, 2009.
Minsky, M. The Emotion Machine, Simon & Schuster, 2006.

Trigo, P. and Coelho, H. Decision Making with Hybrid Models: The Case of Individual and Collective Motivations, Proceedings of the EPIA-07 International Conference (New Trends in Artificial Intelligence), pp. 669-680, Guimarães, 2007.

Trigo, P. and Coelho, H. Decisions with Multiple Simultaneous Goals and Uncertain Causal Effects, in Artificial Intelligence in Theory and Practice II, IFIP Volume 276, Springer-Verlag, pp. 13-22, 2008.

Trigo, P. and Coelho, H. Simulating a Multi-Agent Electricity Market, in Proceedings of the 1st Brazilian Workshop on Social Simulation (BWSS-08/SBIA-08), Bahia, October 26-30, 2008.

Wiegel, V. Building Blocks for Artificial Moral Agents, Proceedings of EthicalALife06 Workshop, 2006.

Development of an Adaptive Interface for the Electronic School Notebook

Luís Alexandre[1], Salvador Abreu[11]

[1]LabSI[2] / ESTIG, Instituto Politécnico de Beja, Portugal

[11]Universidade de Évora and CENTRIA FCT/UNL, Portugal

luis.alexandre@ipbeja.pt, spa@di.uevora.pt

Abstract. In this article we describe the construction of an adaptive interface for the Electronic School Notebook, with the ability to predict the most likely task that a user is about to perform, based on prior knowledge of actions already made in the system, combined with the action time frame. This adaptive capability will reduce the number of explicit interactions to accomplish a certain task. The system makes use of machine learning algorithms, based on decision trees and Markov chains.

Resumo. Neste artigo descrevemos a construção de uma interface adaptativa para o Caderno Escolar Electrónico, com a capacidade de prever a tarefa mais provável que um utilizador irá realizar no sistema. Esta capacidade de predição baseia-se no conhecimento das tarefas realizadas anteriormente pelo utilizador em conjugação com o espaço temporal das mesmas. Esta interface adaptativa permitirá reduzir o número de interacções explícitas com o sistema, com vista à realização de uma tarefa. O sistema utiliza algoritmos de aprendizagem automática, baseados em árvores de decisão e redes de Markov.

Keywords: Interactive systems, Students with Special Needs, Support Technologies, Human Computer-interaction, Machine Learning, Prediction, Adaptation.

1 Introduction

The Electronic School Notebook (CE-e) [1] in Fig. 1, has become a valuable tool for children with special needs, in the organization of notes in a classroom environment and also in the organization, in a logical manner, of electronic documents related to a given course.

The CE-e together with other assistive technologies, like Eugénio [2] actively support these children, allowing them to perform writing tasks within a very similar time period to the one spent by children without special needs. This tool promotes the

adoption of a methodical process of collecting notes in classroom context, as a result of this process, remarkable benefits can be obtained in terms of academic success [4].

If the system becomes aware of the task at hand by the user, it will be able to better support users, by helping in accomplishing the task.

In some situations it's very difficult to determine the intentions behind a user actions sequence, but in other situations using the domain knowledge [5] or observing the users behavior in similar situations [6], we are able to predict the goal of the user. An obvious case is the opening of a new lesson, of a certain course with the combination of the system time and the information of the student's time table.

Not only chidrem with motor impairments can benefit from these technologies, also students with cognitive impairments can work in a more autonomous manner with the help of these systems. These kind of mecanisms have already been tried in other aplications, like email clients [6], in order to free the user from the execution of routine tasks by delegating the execution responsibility to the system. This approach has also been tried in the development of Eugénio's keyboard assistant [2][7].

The project guiding principle is DWIM (Do What I Mean) [3]. This principle states that the system should behave exactly according to the user objective, instead of behaving according to a miss-executed instruction, not intended by the user. This is an unattainable goal; still it's guiding our intentions.

The main objective of this project is to implement and evaluate an adaptive interface for the Electronic School Notebook that allows students with special needs a higher level of support in performing several tasks.

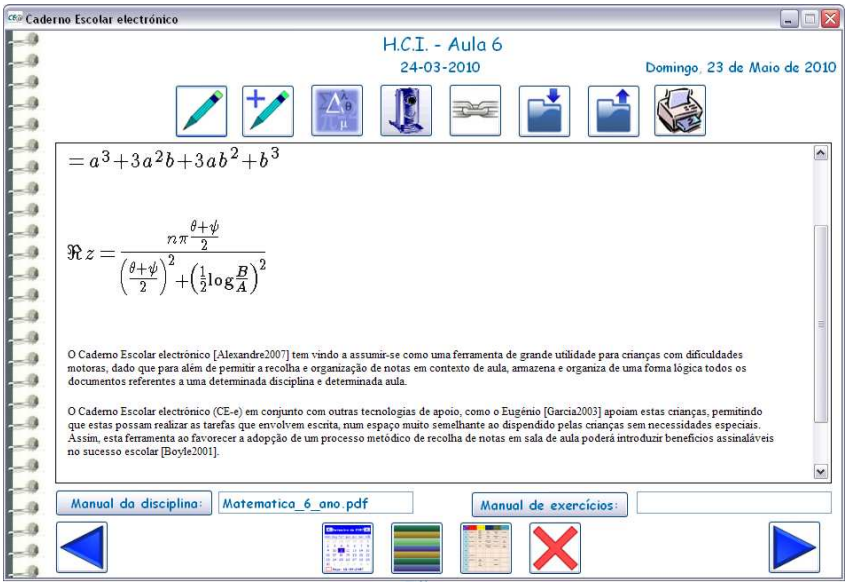


Fig. 1. CE-e notetaking interface

To attain this objective the system must possess application and user models [5] in order to identify the user intentions to provide the necessary assistance in the accomplishment of the task.

In a first moment we gathered the required information to build a knowledge base that contains all the tasks performed by the user on the system, the different actions that lead to the task and finally the context in which they were performed. To characterize the context we gathered a timestamp composed of the date, day of the week, the current time and the associated activities. These records were gathered during normal classes, in the student's personal computers, using CE-e. All the students have cerebral palsy, but this problem only affects them physically. None of these students uses any custom I/O devices, such as switches or adapted mice.

This data and knowledge base was used on the next phase of the project, were we built and evaluate support mechanisms that help users in the accomplishment of tasks, based on the knowledge contained by the knowledge base. This is the main contribution of the project. If the system is able to correctly interpret all the context dimensions, it may evolve from a set of hardcoded rules that provide a limited set of adaptations, to a system that with its continued use will be able to learn and create new adaptations, more appropriate for the user profile. This "intelligence" will only be possible with the use of machine learning algorithms.

On Section 2 of this article we will describe the sate of the art in this subject. The description of the system is made on Section 3 and finally on Section 4 we present a conclusion and propose directions for work to be done in the future.

2 State of the Art

Nowadays we are surrounded by computer systems (e.g. desktops, laptops, PDAs, among many others) and we become computer dependent, all aspects of our lives are determined by decisions or actions that took place on a computer.

However these systems could provide a greater support, if they were able to provide it in an automatic and autonomous form. This support is only possible if the system is able to determine or recognize the context in which both the user and the machine are inserted [8].

Many researchers have reported the importance and benefits that context knowledge can bring to the usability of computer systems thereby increasing and enhancing the Human-Computer Interaction [9]. The main idea behind this knowledge of the context, known as context-awareness, states that computers can feel and react based on the environment. In order to enable this reaction, computer systems should have at their disposal a set of stimuli and prior knowledge that are combined with a set of hardcoded rules. The reaction is the logical result of these calculations [10].

A context-aware system can and should try to make assumptions about the surrounding environment. In [12] the context is defined as "*any information that can be used to characterize a situation of any one entity*".

Context-awareness is often used in satellite tracking systems or in ubiquitous computing. In these systems it's possible to use sensors and other systems that capture with precision part of the context, and thus can, for example, propose and make environmental changes (e.g. environment light or open the windows in buildings, to minimize energy consumption).

Initially the context was understood has being the location, but more recently some researchers proposed that the location coulnd't be understood as the context of a situation, but rather as a component of context [12]. It was further proposed that the context could be used to build adaptive and intelligent interfaces, in which the interaction sould be as implicit as possible.

Many of the studies and attempts of adaption to context have passed through the use of satellite tracking devices. These systems use only a small component of a relatively large universe of possible adaptations to the context. A system that only uses the location to fit the context, can not be considered a context-aware system, and should be considered as location-aware system [13].

In the last years, context information has been combined with time; as result of this many researchers have proposed several types of "intelligent" interfaces (intelligent Human-computer Interaction).

Examples of this new kind of interaction are the "Adaptive and Predictive Interfaces". These interfaces try to minimize the two major problems that affect the interaction between humans and computers: (i) The considerable quantity of information and widgets that populate the interfaces; and (ii) The fact that users must decide the next step or action in a very short time [15].

The idea behind these interfaces is the following: "*Instead of being the user to adapt to the system, the system adapts to the user*". Although based on a known premise, the inherent problems are in large number and complexity. An adaptive interface must be based on cognitive models and on the surrounding environment [15]. These models try to explain the user's level of expertise and experience, through parameters such as: (i) commands made, (ii) error rates, (iii) typing speed, among others. An adaptive interface can make the adjustments in one of two forms:

- The adaptation responsibility is splited between the system and the user, for example, the system makes a small adaptation and waits for its acceptance, if the adaptation is not accepted or not satisfactory, the system makes a new adaptation;
- Alternatively, the system suggests the most adequate adaptation for the current context in a totally independent mode.

The second form is considered by many has the true adaptability. However, this kind of adaptation is very difficult to achieve and it's only possible with the use of machine learning algorithms. The concept behind machine learning can be considered as adaptive, because the algorithms of this type base their decisions on prior information (train and test sets).

However an adaptive interface also has its problems and among others we can mention the fact that the user can not create a mental model of the system, if the appearance of the system often changes. Another potential problem may result from

the loss of control that a user may feel. Finally, and perhaps the biggest problem of adaptive interfaces, turns out to be the cost and complexity of the implementation.

Another paradigm of adaptability and predictability derives from the attention and suggestion. In this case the interfaces are alert and automatically suggest actions to users.

An attentive interface automatically sets priorities and in accordance with these priorities presents to the user the most appropriate information or options regarding the current context and time, in order to optimize the resources of both user and system. With this optimization, the interaction actors never incur in overloading [16]. As in adaptive interfaces, attentive interfaces base their decisions on information and models, based on decisions previously made by the user.

On the other hand we have the suggestive interfaces, which only provide the user with clues about the next possible action, through the enhancement of certain interface components [17].

The interaction with a suggestive interface is very simple, in fact the user only has to choose one of the highlighted options or if none of the options matches the desired one, the user only has to choose the desired option. Usually suggestions are generated by a system often called "suggestions engine", which constantly observes the system state, generating a set of suggestions that matches the patterns closest to the current state. A suggestive interface can be viewed as a gestures interaction system, i.e., instead of updating the system automatically, when it discovered a similar pattern, the interface presents a set of hints and prompts the user to choose one of these.

Suggestive interfaces are an extension of predictive interfaces and are being, viewed by many researchers, as the interfaces of the future, for the vast majority of systems, including WIMP systems (Windows, Icons, Menus and Pointing Devices).

3 Architecture of the Adaptive Interface

The interface of the CE-e prototype was been designed according to a design methodology for interactive systems with focus on the user and, therefore is quite simple and intuitive [1]. However as CE-e intends to provide as much help as possible in tasks of notetaking and information organization, it's intended that the interface can alleviate, as much as possible, the user of routine and repetitive tasks, which may be particularly useful for users with severe motor problems. Thus, based on our experience, in the opinion of technicians and other experts, we decided to implement a suggestive interface on CE-e.

We made this choice because these kind of interfaces are seen as a possible path for the future of interfaces, since they exploit context-awareness in order to reduce the number of explicit commands to be performed by users, required for any given operation [17][25]. The adaptive interface, designed for the CE-e, meets all these requirements, extending the traditional notions of an interface.

cid	name	type	notnull	dflt_value	pk
0	ID	INTEGER	0		1
1	date	NVARCHAR(40)	0		0
2	weekDay	NVARCHAR(20)	0		0
3	time	NVARCHAR(50)	0		0
4	form	NVARCHAR(50)	0		0
5	discipline	NVARCHAR(50)	0		0
6	event	NVARCHAR(50)	0		0

Fig. 2. CE-e log structure

3.1 The Knowledge Base

As previously stated, the system's ability to anticipate or suggest the most probable action for a certain context, can only happen if the system has access to a knowledge base with the knowledge acquired in prior interactions with the system. To build this knowledge base researchers often use a technique called "logging".

For some years, researchers have been working in Augmentative and Alternative Communication (AAC) systems. These efforts are only justified if the impact of these technological advances can be measured. Thus, in order to facilitate the analysis of the collected data; researchers have defined a standard logging format that allows the analysis of data in a systematic way, called Universal Logging Format [18].

The log file structure proposed in [18], has three distinct parts: (i) The head, which specifies the content and format of records; (ii) Body, which is composed of *n* lines, each representative data on a log; (iii) Analysis section (optional), where some statistical analysis of the logs can be referenced.

The logs have a time component, an output or result (action), the type of device used which originated the log, type of communication that was being developed, the context (in case of a AAC tool), which words preceded the log and an indication of, where in the system, the action was triggered.

Under this format, we built a log system with the structure presented in Fig. 2 that contains the following fields: date, day of week, time, system page, active discipline and resulting action.

A log mechanism it's useful if we have a fast, precise and powerful form to extract the information we want. Inspired in [19] and on features of relational DBMS we decide to use SQLite[1]. This relational database management system, besides allowing the construction of queries in standard SQL, doesn't need to have an active server in the system. This log mechanism, adds to the capabilities of SQL, a total independence of the system, since this RDMS is added to the application via a DLL built in C Programming Language.

[1] Project Web site: http://www.sqlite.org/

ID	date	weekDay	time	form	discipline	event
1	26-02-2010	Friday	11:9:58	FormAgenda		iniciar_aplicacao
2	26-02-2010	Friday	11:10:56	FormAgenda		vista_de_livros
3	26-02-2010	Friday	11:11:6	BooksForm		abrir_caderno_Matemática
4	26-02-2010	Friday	11:11:14	NotesForm	Matemática	vista_livros
5	26-02-2010	Friday	11:11:16	BooksForm		abrir_caderno_Língua Portuguesa
6	26-02-2010	Friday	11:11:20	NotesForm	Língua Portuguesa	aceder_manual_disciplina
7	26-02-2010	Friday	11:12:21	NotesForm	Língua Portuguesa	aceder_manual_exercicios
8	26-02-2010	Friday	14:6:56	FormAgenda		iniciar_aplicacao
9	26-02-2010	Friday	14:7:11	FormAgenda		vista_de_livros
10	26-02-2010	Friday	14:7:16	BooksForm		abrir_caderno_Estudo do Meio
11	26-02-2010	Friday	14:7:22	NotesForm	Estudo do Meio	aceder_manual_disciplina
12	26-02-2010	Friday	14:56:21	NotesForm	Estudo do Meio	terminar_aplicacao
13	02-03-2010	Tuesday	9:18:29	FormAgenda		iniciar_aplicacao
14	02-03-2010	Tuesday	9:18:38	FormAgenda		vista_de_livros
15	02-03-2010	Tuesday	9:18:44	BooksForm		abrir_caderno_Matemática
16	02-03-2010	Tuesday	9:18:50	NotesForm	Matemática	nova_nota_de_texto
17	02-03-2010	Tuesday	9:26:49	NotesForm	Matemática	vista_livros
18	02-03-2010	Tuesday	9:26:53	BooksForm		abrir_caderno_Língua Portuguesa

Fig. 3. Extract from a log file

3.2 The Algorithms

For some time now, that researchers in the field of Human-computer Interaction, develop studies on the adaptability of interfaces to users, through learning systems based on machine learning, both for traditional desktop interfaces and for Web interfaces [14][15][20].

Before the implementation of the system, the logs were subjected to machine learning tools, to check which classification algorithm could offer better results.

We can divide machine learning, through classification, in two major groups: Supervised Learning and Unsupervised Learning [21]. The collected logs for the formation of the knowledge base are constituted by the time frame, context and the performed action, i.e. the user's objective. In this scenario we face a case of supervised learning, where the user actions will be the class. Supervised learning can be further divided in classification or regression problems. Since our aim is the prediction of the user most probable action, in a multiclass environment, we have a classification problem.

In the possession of real use records (Fig. 3), it was possible to start working in the "intelligence" component of the system, in order to prove the feasibility of an intelligent interface for the Electronic School Notebook. This type of tests is known as "proof of concept".

The Weka environment [22] was chosen for the knowledge base tests in search of patterns that could justify the system implementation. This environment provides implementations of the majority of machine learning algorithms, like decision trees. Our first choice fell on Decision-trees, since the system will work in real time, making queries to the knowledge base and presenting the hypothesis that best fits the context of the moment.

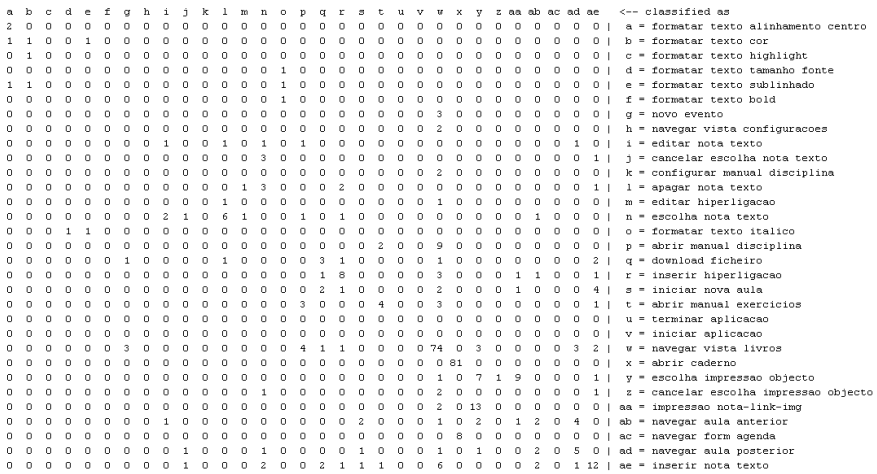


Fig. 4. Confusion matrix generated by Weka with the data from the knowledge base

The first tests carried out, using a C4.5 decision tree [23] showed that this machine learning algorithm can reach a hit rate of 50%, which justifies the development and implementation effort, since the system has forty classes. However these forty options aren't always visible, depending on the screen where the user is located, the system will only present between three and seventeen options.

Other classification algorithms were tested, searching for better results or outcomes that strengthen those obtained using decision trees, particularly Markov chains and Hidden Markov Models [26].

The inference engine of the system will have to manage a small set of attributes, as shown in Fig. 2. The engine will only have four attributes that need to be analyzed, as the fifth attribute is the class attribute, which will serve to train the classifier. However, this classifier will have to deal with a multiclass problem with a large number of classes. The confusion matrix build by Weka (Fig. 4), clearly shows the number of classes. This particularly large number of classes is a possible problem for the classifier, because the resulting tree will have a large number of nodes and leaves, thereby increasing the probability of errors in the suggestion of the most likely action. However the 52% accuracy achieved, ensure that at least one in every two suggestions is correct, thus reducing the work load on the user, necessary to perform tasks that add value to the work of the user in the system.

3.3 The Encoding

The inference engine encoding is being developed with two algorithms, a C5 decision tree, which derives from the C4.5 tree [23] and a Markov chains based algorithm. The decision tree is fastest and more efficient in the process of finding patterns and in the classification of new samples. The implementation with a Markov chains algorithm follows a different methodology. While the decision tree is being implemented with a

DLL based on open-source code, the second is hardcoded and integrated in the source-code of CE-e.

The ultimate objective of the project is the construction of a prototype using the C5 decision tree, but the encoding process of the DLL is delayed due to some problems.

As soon as possible, we wanted a prototype that demonstrates the usefulness of the Electronic School Notebook adaptive interface, so we decided to implement the inference engine of the CE-e with a Markov chains based algorithm. The final version of the system prototype will not be available with this algorithm, because the tests results were poorer, in terms of hits and speed.

After the proof of concept has been held, the DLL based on the decision tree [23], will be developed. This effort is justified by the fact that the DLL is an independent and portable component, allowing and ensuring its reuse in other systems. Another fact that justifies this effort is that the development of new versions of CE-e, with and without an adaptive interface, will be simpler and we don't have the necessity of building two incompatible versions.

With the introduction of the inference engine in the CE-e prototype, shown in Fig. 5, the system had to undergo some changes, so that the various components of the interface could be dynamically changed by the interface inference engine.

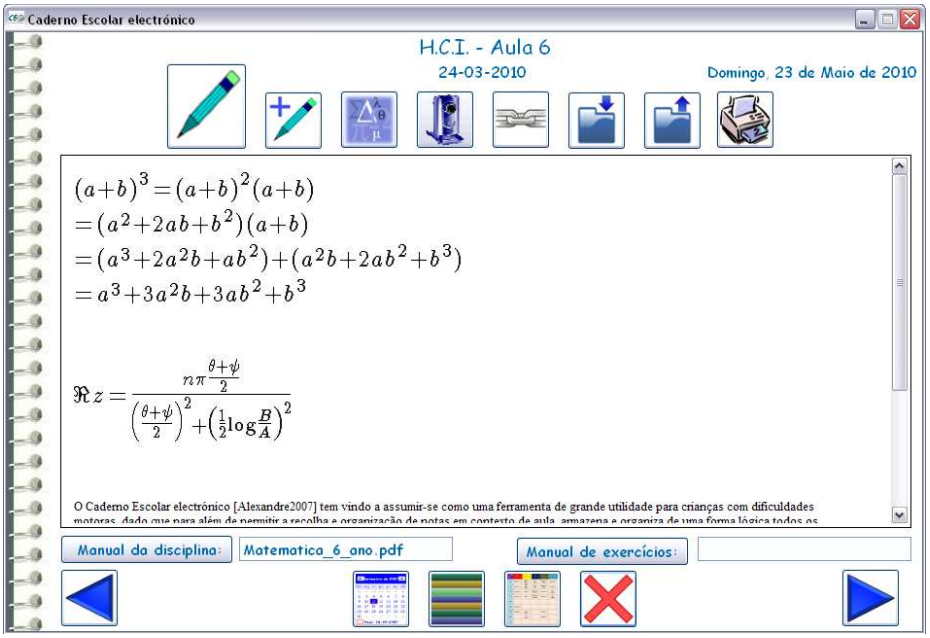


Fig. 5. System prototype with the inference engine

4 Conclusions and Future Work

In this article we propose and present an adaptive interface for a tool that aims to be an alternative to traditional school notebooks, for students with special needs, particularly those who only have physical problems, that restricts the mobility and ability to manipulate traditional I/O mechanisms.

Currently we have a prototype of the CE-e with the adaptive interface in the final stages of development. This initial prototype will only be used for testing and proof of concept, which are ongoing.

The interface makes suggestions to the user using the knowledge base, which contains information about past actions in the system. The analysis is carried out using techniques of machine learning and algorithms based on Markov chains.

With the objective of building a knowledge base that could answer the questions asked during the analysis and decision phase, a relational database log mechanism was built, using the SQLite database manager.

We want a system that continuously learns. This learning process can quickly push the number of lines, of the log file, to a few thousand. As a result, the inference engine can become very slow. A possible solution to this problem is the limitation of the log file, in number of lines or size. This limitation will eliminate automatically the logs with a certain timestamp.

However, the assumption of a temporal limitation for the logs can still change as we are yet analyzing which option will gives a higher percentage of hits. If the system only has access to the logs of the last thirty days of use, will have a temporally limited knowledge, but this knowledge may focus on options that the user has used more frequently. Moreover, and assuming that the logs can grow indefinitely, the system will have an overview of all actions and choices carried out by the user, which could bring higher probabilities of success. This decision can only be taken, after a period of use and testing with a duration not smaller than six months. These tests will determine what is the best structure for the knowledge base, "with forgetfulness" or "without forgetfulness".

A possible solution for the continuous collecting of logs is an offline processing, which will lead to the construction of an index of actions. This technique is quite used in information retrieval systems [24].

Another inherent objective of this project is to provide the system with a set of hardcoded rules, which in case no correspondence between the rule and the result obtained by inference engine, the rule will override the inference engine decision. As an example of such rules we have the timetable that can provide information about the most likely task to execute on the system, according to the time component provided by the operating system.

The suggestion mechanism highlights the most likely option. This enhancement is combined with a keyboard shortcut, to avoid a mouse movement to confirm the option. As earlier mentioned, a suggestive interface can be viewed like a pseudo-gestural interface, because instead of immediately responding to the users input, updating the interface immediately, the system first asks the confirmation after showing multiple suggestions. On CE-e the inference engine will only highlight one

option, and the user can complete the action by choosing the presented option, or can ignore it, by choosing another option, rather than the one presented.

CE-e wants to be an effective alternative to traditional school notebooks. If the introduction of an adaptive interface proves to be an asset, students with physical and motor difficulties, will have at their disposal a tool that will help them further more in the organization of all writing tasks and in the organization of electronic documents.

By now, the adaptive CE-e has an interface that can predict the most likely action for a certain context, which reinforces the lemma of the project: Help and relieve students with special needs of routine tasks, that don't add capital gains to the work that they have to perform, in order to assist in their integration into regular education.

The final testing phase is in progress. These tests will bring the answers regarding the effectiveness of the suggestive interface, designed and built for CE-e.

Acknowledgements

We would like to thank the valuable collaboration, inputs and ideas of the faculty staff of the Polytechnic Institute of Beja assigned to the Information Systems and Interactivity Lab (LabSI^2).

References

1. Alexandre, L., Garcia, L., Bruno, L.: Development of an electronic scholar notebook for students with special needs. In DSAI2007. Vila Real, Portugal (2007).
2. Garcia, L.: Concepção, implementação e teste de um sistema de apoio à comunicação aumentativa e alternativa para o português europeu. Tese de Mestrado. Universidade Técnica de Lisboa, Instituto Superior Técnico (2003).
3. Teitelman, W.: A tour through cedar. Proceedings of the 7th international conference on Software engineering. Pages: 181 – 195. Orlando, Florida, United States (1984).
4. Boyle, J.R., Weishaar, M.: The effects of strategic notetaking on the recall and comprehension of lecture information for high school students with learning disabilities. Learning Disabilities Research & Practice, 16(3), pp. 133–141, (2001).
5. Dix, A., Finlay, F., Abowd, G., Beale, R.: Human Computer Interaction, 3rd Edition. Prentice Hall (2004).
6. Maes, P.: Agents that reduce work and information overload. Communications of the ACM 37(7) (1994).
7. Rodrigues, N.: Desenvolvimento de mecanismo de interacção preditiva para aumentar o desempenho de tarefas. Projecto de Licenciatura. Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Beja (2007).
8. Blum, M.L.: Real-time context recognition. Master's thesis, Department of Information Technology and Electrical Engineering, Swiss Federal Institute of Technology Zurich - ETH (2005).

9. Bradley, N.A., Dunlop, M.D.: Toward a multidisciplinary model of context to support context-aware computing. Human Compute-Interaction, 20:403 – 446, (2005).
10. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. IEEE Workshop on Mobile Computing Systems and Applications - WMCSA'94, 1:89 – 101 (1994).
11. Schilit, B.N., Theimer, M.M.: Disseminating active map information to mobile hosts. IEEE Network, pages 22 – 32 (1994).
12. Dey, A.K.: Understanding and using context. Personal Ubiquitous Computing, 51:4 – 7 (2001).
13. Barnard, L., Yi, J.S., Jacko, J.A., Sears, A.: Capturing the effects of context on human performance in mobile computing systems. Personal and Ubiquitous Computing, Volume 11, Issue 2, Pages: 81 – 96. Springer-Verlag (2006).
14. Langley, L.: Machine Learning for Adaptive User Interfaces. Proceedings of the 21st Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence. Pages: 53 – 62 (1997).
15. Norcio, A.F., Stanley, J.: Adaptive human- computer interfaces: A literature survey and perspective. IEEE Transactions on Systems, Man and Cybernetics, 19, No.2:399 – 408 (1989).
16. Vertegaal. R.: Designing Attentive Interfaces. ETRA'02. N.O., USA (2002).
17. Igarashi, T., Hughes, J.F.: A suggestive interface for 3D drawing**.** Proceedings of the 14th annual ACM symposium on User interface software and technology. Pages: 173 – 181. Orlando, Florida (2001).
18. Lesher, G.W., Moulton, B.J., Rinkus, G., Higginbotham, D.J.: A Universal Logging Format for Augmentative Communication. Enkidu Research, Inc. Lockport, NY (2000).
19. Gonçalves, M.A., Panchanathan, G., Ravindranathan, U., Krowne, A., Fox, E.A.: JCDL'03 Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries (2003).
20. Frias-Martinez, E., Chen, S.Y., Liu, X.: Survey of Data Mining Approaches to User Modeling for Adaptive Hypermedia. IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, Volume 36, No. 6 (2006).
21. Alpaydin, E.: Introduction to Machine Learning. MIT Press (2004).
22. Garner, S.R.: Weka: The Waikato environment for knowledge analysis. In Proc New Zealand Computer Science Research Students Conference, pages 57-64, University of Waikato. Hamilton, New Zealand (1995).
23. Quinlan, R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers (1993).
24. Manning, C.D., Raghavan, P., Schütze, H. An Introduction to Information Retrieval. Cambridge University Press (2008).
25. Van Dam, A. Post-WIMP User Interfaces, Communications of the ACM, Vol. 40, No. 2, pp. 63-67 (1997).
26. Rabiner, Lawrence R. (February 1989). A tutorial on Hidden Markov Models and selected applications in speech recognition". Proceedings of the IEEE **77** (2): pp. 257–286 (1989).

Jogos de Papéis e Emoções em Ambientes Assistidos

Luis Machado, Davide Carneiro, Cesar Analide, and Paulo Novais

Departamento de Informática, Universidade do Minho,
Braga, Portugal
luisp_machado@hotmail.com,{dcarneiro,analide,pjon}@di.uminho.pt

Resumo Projectos de cenários dotados de Inteligência Ambiente têm vindo a crescer de dia para dia e, mesmo sendo uma área relativamente recente, já mostram a sua importância. Com esta tecnologia, é possível encontrar soluções para áreas tão vastas como a medicina, o entretenimento e lazer ou a segurança, procurando, sempre, o bem-estar e a segurança dos seus utilizadores. Neste artigo faz-se um levantamento sobre as principais técnicas de Jogos de Papéis e Computação Afectiva, mostrando o seu importante papel em cenários dotados de Inteligência Ambiente. Apresenta-se ainda o trabalho realizado na extensão de uma plataforma de simulação de Ambientes Inteligentes com conceitos de computação afectiva, nomeadamente, acrescentando a capacidade de lidar e de reagir a informação que define sentimentos e emoções de utilizadores.

Palavras-Chave: Computação Afectiva, Jogos de Papéis, Emoções, Inteligência Ambiente, Ambientes Assistidos.

1 Introdução

A Inteligência Ambiente é definida pela *IST Advisory Group (ISTAG)* como ambientes que são sensíveis e respondem à presença de pessoas. É possível encontrar nesta tecnologia soluções para áreas tão vastas como a medicina, o entretenimento e lazer ou a segurança. É objectivo deste tipo de ambientes terem comportamentos pró-activos coerentes, que zelem pelo bem-estar e segurança dos seus utilizadores. São ainda caracterizados por terem a capacidade de prestar assistência nas tarefas do dia-a-dia de quem os utiliza.

A Inteligência Ambiente apoia-se essencialmente em três tecnologias: redes de computadores, computação ubíqua [8] e interfaces inteligentes [32]. No paradigma da computação ubíqua, o poder computacional encontra-se distribuído por dispositivos electrónicos cada vez mais pequenos, que, muitas vezes, passam despercebidos ao utilizador. Os interfaces inteligentes apresentam uma nova forma de interacção Homem-Máquina, mais intuitivos e mais fáceis de utilizar que se libertam dos mecanismos e limitações que até hoje os interfaces tradicionais têm mostrado. São, ainda, caracterizados por usarem mecanismos de interacção mais naturais para o Homem, como, por exemplo, a voz, os gestos, ou, até, o estado de espírito.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 745–756

1.1 Inteligência Ambiente

A Inteligência Ambiente (IAm) [31] [29] é um paradigma computacional relativamente novo na sociedade da informação, no qual as capacidades das pessoas são aumentadas através de um ambiente digital que é "consciente"da sua presença e contexto [1], sendo sensível, adaptativo e atento às suas necessidades, hábitos e emoções. A IAm pode ser definida como a junção da computação ubíqua com os interfaces inteligentes. É constituída por uma rede de dispositivos predominantemente móveis. Ao adicionarmos métodos adaptativos de interacção com o utilizador, o resultado são ambientes digitais criados para melhorar a qualidade de vida das pessoas, pela tomada de decisão autónoma [4]. Estes sistemas, conscientes do contexto, combinam informação ubíqua, comunicação e entretenimento, com personalização, interacção natural e inteligência. O caminho a seguir para atingir este objectivo recorre a áreas tão diferentes como a inteligência artificial, a psicologia, a lógica matemática, bem como diferentes paradigmas computacionais e metodologias de resolução de problemas, como, por exemplo, mecanismos de Suporte à Decisão em Grupo.

1.2 Jogos de Papéis

Os Jogos de Papéis (Role-Playing Games - RPG) consistem num tipo de jogo onde os jogadores "interpretam"uma personagem, criada dentro de um determinado cenário (ambiente). As personagens respeitam um sistema de regras, que servem para organizar as suas acções, determinando os limites do que pode ou não ser feito [20]. RPG é uma técnica muito utilizada para treinos, porque permite colocar os jogadores frente a situações de tomada de decisão sem implicar enfrentar as consequências reais. Grandes empresas têm utilizado RPG em cursos de formação devido ao factor lúdico envolvido nos jogos, o que faz com que o treino e/ou aprendizagem de determinado assunto seja facilitado.

Desta maneira, são jogos onde cada jogador desempenha um papel e toma decisões, a fim de alcançar os seus objectivos. Na verdade, os jogadores utilizam RPG como um "laboratório social", isto é, como uma forma de experimentar uma variedade de possibilidades, sem sofrer as consequências do mundo real [6].

1.3 Emoções

Uma das primeiras abordagens ao estudo sistemático das emoções foi realizada por William James [19] e Carl Lange [22], que, embora tenham desenvolvido os seus estudos separadamente, anunciaram os seus resultados aproximadamente ao mesmo tempo, o que fez com que a teoria que anunciavam ficasse conhecida como a de James-Lange. Esta teoria sugere que as emoções são consequência de uma resposta fisiológica dos humanos a estímulos externos, sendo identificadas através da análise às respostas dadas por estes (i.e., se vejo um urso, então fico a tremer, se estou a tremer então estou com medo, onde medo é a emoção identificada). De acordo com esta teoria, a cada emoção está associada uma reacção fisiológica diferente [16].

Muitos autores criaram outras teorias à sua volta, como é o caso de Walter Cannon [9] e Philip Bard [5] (segundo esta última teoria, quando uma pessoa chora é porque está triste, enquanto que na teoria de James-Lange, a pessoa está triste porque chora). Mais tarde, o trabalho de Madga Arnold deu origem a muitas outras teorias de avaliação, sendo de realçar os trabalhos de Nico Frijda [14] e de Richard Lazarus [23] [24].

1.4 Plataforma VirtualECare

O projecto VirtualECare [11] está a ser utilizado como caso de estudo. O seu principal objectivo é o de desenvolver um sistema multi-agente inteligente capaz de monitorizar, interagir e fornecer aos utilizadores serviços de saúde de qualidade melhorada. Este sistema irá ser interligado, não apenas com outras instituições de prestação de cuidados de saúde, mas, também, com centros de lazer, estruturas de formação, lojas ou até mesmo os parentes do paciente, para citar alguns exemplos.

A arquitectura do VirtualECare (Figura 1) é uma arquitectura distribuída com os seus diferentes módulos interligados através de uma rede (p.e. LAN, MAN, WAN). Os módulos são: *Supported User (a)*, *Environment*, *Group Decision (b)*, *CallServiceCenter (c)*, *CallCareCenter (d)*, *Relatives (e)*. Cada um destes módulos desempenha um papel específico [28].

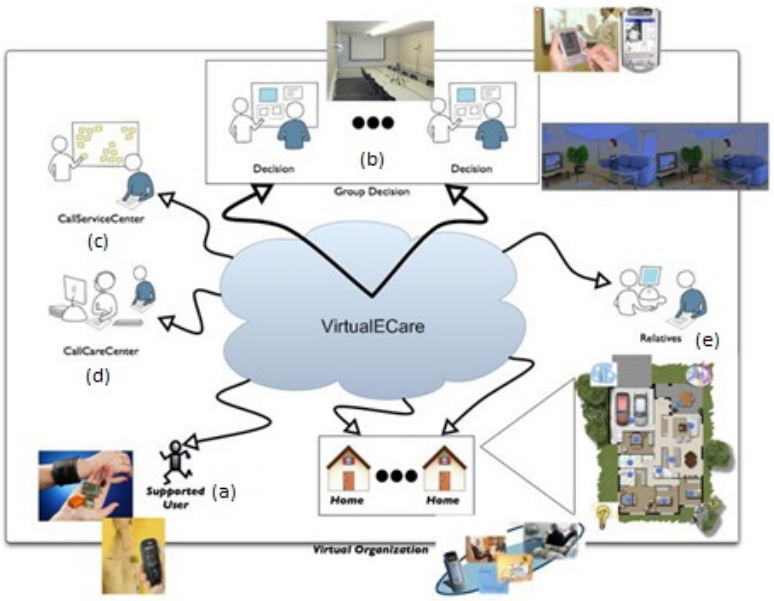


Figura 1. Arquitectura da Plataforma VirtualECare [11]

2 Jogos de Papéis

Os Jogos de Papéis, mais conhecidos por RPG (Role-Playing Games) consistem numa categoria à parte nos jogos, optando pela colaboração em vez da competição: os RPG não são jogos com um final com ganho ou perda. No final, deve completar-se uma história construída a partir das regras do jogo, procurando objectivos individuais e/ou colectivos. Além disso, é um jogo em que o discurso, diálogo e troca de ideias são vitais para o seu desenvolvimento.

Os RPG possuem o potencial de, através do exercício da fantasia, agir positivamente no desenvolvimento mental do homem e, consequentemente, no seu desenvolvimento social. Se observado com maior cuidado, pode-se perceber a força de integração latente de auxílio pedagógico, pois o jogo estimula uma troca constante de informações e experiências. Assim, "se bem direccionado e explorado, o RPG tem tudo para ter um papel marcante na sociedade"[20].

2.1 Jogos de Papéis e a Socialização

Os RPG destacam-se por ter a fantasia como principal instrumento. O jogador tem a oportunidade de viver diferentes personagens, viver em diferentes mundos, em diferentes realidades. E é isso que faz dele um jogo com possibilidades incomuns. Segundo Freud [13], "A fantasia é fundamental para o desenvolvimento do pensamento, para o relacionamento do homem com a realidade". Os RPG permitem ao jogador exercitar a sua fantasia e torná-la aceitável no seu meio, o que confere ao jogo o papel de elemento social. No momento em que o jogador começa a viver a sua personagem na história e a sentir-se aceite, as suas inibições são despidas, e isto favorecerá certamente a sua socialização.

A capacidade de integração do RPG começa na própria estrutura do jogo: é jogado em grupo, sendo que não é voltado para a competição, mas sim para a cooperação entre seus participantes [3].

Os grupos de RPG acabam por ser construídos em torno das suas afinidades. Geralmente, um grupo de RPG costuma ouvir o mesmo tipo de música, filme, ou ter um conjunto de referências mais ou menos similares.

Dentro de uma sociedade que se mostra cada vez mais complexa devido, por exemplo, ao desenvolvimento tecnológico, não será exagero supor que o jogador de RPG está, à partida, mais apto para agir nesta sociedade [3].

2.2 Jogos de Papéis em cenários dotados de inteligência ambiente

Focando o conceito dos Jogos de Papéis, onde cada jogador pode encarnar uma determinada personagem, num cenário dotado de inteligência ambiente podem ser criadas várias personagens-tipo com as respectivas regras associadas a cada uma delas. Aqui, cada utilizador assume determinado papel e é reconhecido pelo sistema como tendo determinadas características e determinadas regras que aquele papel lhe permite usufruir. Assim, é possível simular, com a proximidade à realidade que o sistema pretender, isto é, podem ser criadas tantas personagens quantas as necessárias para corresponder o melhor possível à realidade de um ambiente do género.

2.3 Trabalhos relacionados

Muitos autores relacionam as técnicas dos RPG com sistemas computacionais devido ao seu factor lúdico. Nesta secção, são apresentados alguns destes trabalhos, onde se pode verificar a existência de uma grande diversidade. Parte deles são sistemas educacionais em que o utilizador pode testar os seus conhecimentos, perceber quais as suas lacunas e aprender com o sistema. Um desses exemplos é apresentado no artigo *"O Desenvolvimento de um Protótipo de Sistema Especialista Baseado em Técnicas de RPG para o Ensino de Matemática"* [36], onde se apresenta um modelo computacional baseado em técnicas de Sistemas Periciais e de RPG, que permite ao utilizador, ao exercitar a sua fantasia, testar os seus conhecimentos matemáticos. G.Schlup, et al, autores de *"RPG Educacional Utilizando o Conceito de Agentes"* abordam, também, esta temática ([33]).

No artigo *"A Computer-based Role-Playing Game for Participatory Management of Protected Areas: The SimParc Project"* [18] é apresentado um exemplo de como os RPG podem ser usados para dois efeitos complementares: na ajuda e extracção de experiências sociais, e, também, no apoio à tomada de decisão. É explorado no contexto da gestão participativa de áreas protegidas para a conservação da biodiversidade e inclusão social.

Elisa Mattarelli et al, autores do artigo ''Design of a role-playing game to study the trajectories of health care workers in an operating room"[25] através dos jogos de papéis, aliados às técnicas de simulação, tentam prever ou simular os mecanismos de coordenação dentro de blocos operatórios, e, outros ainda que apresentam arquitecturas genéricas para suportar o processo de negociação num cenário de resolução de conflitos [2].

Por fim, e ainda relacionado com os jogos de papéis, encontramos o projecto *KidZania*[1]. Este projecto oferece um parque temático dirigido às crianças, onde estas podem "brincar aos adultos"num ambiente altamente realista.

3 Emoções e Computação Afectiva

Até finais do século XX as ciências consideravam que a emoção existia à parte do raciocínio consciente. Esta noção sofreu mudanças significativas, com estudos oriundos da Neurologia e das Ciências Cognitivas. Entretanto, resultados recentes [12] apontam uma forte ligação das emoções com quase todos os aspectos da cognição e com a origem do pensamento consciente na criança. Este novo entendimento das relações entre emoção e cognição começou a influenciar alguns projectos de sistemas computacionais, em geral, e a pesquisa e o desenvolvimento de sistemas de aprendizagem baseados no computador, em particular.

Estas iniciativas foram agregadas numa sub-área científica da IA denominada por Computação Afectiva (CA). Esta área consiste num conjunto de técnicas adaptadas da IA e da Engenharia de *Software*, agregadas e coordenadas conjuntamente ao estudo, modelação e simulação da experiência afectiva humana, orientado a aplicações nos mais variados domínios [7].

[1] www.kidzania.pt

3.1 Emoção e Inteligência Artificial

Inspirados em modelos psicológicos de emoção, investigadores da área científica da IA começam a reconhecer a importância da modelação da componente emocional quando se trata de desenvolver sistemas computacionais direccionados para a tomada de decisão. Rosalind Picard [21] sintetizou motivações para dotar as "máquinas"de capacidades emocionais, nomeadamente:

- As emoções podem ser úteis na construção de robôs e personagens sintéticas com capacidade de simular o comportamento de seres vivos. O recurso à problemática da emoção aumenta a credibilidade destes agentes perante os seres humanos;
- A capacidade de exprimir e entender a emoção será útil para o melhoramento da interacção Homem-Máquina. Se pensarmos, por exemplo, numa aplicação educacional, será útil se o agente tiver a capacidade de interpretar o estado emocional do utilizador (i.e., através de expressões faciais, pressão sanguínea). Não é de excluir que um utilizador fatigado possa não aceitar determinados tipos de interacções;
- Para construir máquinas "inteligentes"(embora o conceito de "máquina inteligente"não seja bem definido);
- Para entender a emoção e simulá-la. Este é um ponto importante para este trabalho, porque, embora não se pretenda enveredar por um estudo aprofundado deste tema, pretende-se simular o comportamento de um sistema, onde actuam diferentes utilizadores com as suas respectivas características emocionais.

As publicações de Aaron Sloman [35] e de Marvin Minsky [26] foram cruciais para o despertar do interesse dos investigadores da área da IA por esta faceta do comportamento humano.

3.2 Computação Afectiva em cenários dotados de inteligência ambiente

Os cenários dotados de inteligência ambiente pretendem ser, cada vez mais, ambientes capazes de reagir e agir pro-activamente face às necessidades dos seus utilizadores. Para isto, estes ambientes necessitam de perceber, com a melhor fiabilidade, as reais necessidades ou desejos dos seus utilizadores. Neste sentido, a computação afectiva e em particular o reconhecimento das emoções, têm-se revelado de grande importância. Através do reconhecimento das emoções, estes sistemas serão capazes de identificar, com maior precisão, as necessidades do utilizador. Por exemplo, se o sistema detectar que o utilizador se encontra stressado, pode adaptar o ambiente de maneira a acalmá-lo, ligando música calma num volume baixo, em conformidade com as preferências do utilizador.

3.3 Trabalhos relacionados

Na área da detecção e reconhecimento de emoções, existem alguns trabalhos muito relevantes. Estes trabalhos focam, essencialmente, mecanismos de detecção (expressões faciais, batimento cardíaco) e mecanismos de reconhecimento de emoções. O trabalho de A.Herbon et a [17]l, é um exemplo de como medir e reconhecer diferentes estados afectivos (raiva, desgosto, medo, alegria, tristeza e surpresa) dos utilizadores. Estes estados afectivos são reconhecidos através dos batimentos cardíacos do utilizador, a sua actividade electro dérmica, a actividade dos músculos faciais e a sua voz. Existem outros projectos que se preocupam com a comunicação através das emoções. Um desses exemplos é o projecto *Shoji* [34] onde foi desenvolvida uma ferramenta de comunicação que permite enviar e receber informação ambiental como temperatura, luminosidade, nível de ruído, informação da presença ou ausência de indivíduos, os seus movimentos e as suas emoções. Outro exemplo é o projecto *Emotion-Sensitive Robots* [30] que apresenta uma plataforma de cooperação onde o robô é sensível às emoções expressas pelo humano, trabalhando com ele e sendo capaz de mudar o seu comportamento de acordo com a sua percepção. Ainda outro exemplo é o projecto *Oxygen* [2] que visa a comunicação centrada no utilizador e aposta na computação pervasiva através da combinação de uma interacção perceptiva (voz e visão) das necessidades do utilizador, do conhecimento individualizado, de agentes de *software* e de tecnologias de colaboração.

Com uma ideia ligeiramente diferente, Kiel Gilleade et al [15], apresenta a filosofia de manter os jogadores entusiasmados enquanto jogam.

4 Simulação

A simulação é uma operação fundamental quando está em causa a criação de cenários reais onde a margem de erro tem que ser mínima a fim de evitar danos maiores ou mesmo irreversíveis.

Deste modo, uma vez que o cenário que nos propomos trabalhar pode pôr em risco a qualidade de vida de quem o utiliza e confia nele, é prioritária a criação de um sistema de simulação de todo este ambiente, assim como tentar reproduzir todas as falhas possíveis do mesmo. Como já referido anteriormente, é usado como caso de estudo o projecto VirtualECare.

Inicialmente, o projecto VirtualECare considerava apenas um utilizador. O sistema reage e aprende (através de técnicas de *Case Base Reasoning - CBR*) as suas preferências, necessidades e acções. Por exemplo, quando a temperatura dentro do quarto está alta, o utilizador pode optar por baixar as persianas ou ligar o ar condicionado [10]. Este é o tipo de cenário que é possível simular na plataforma VirtualECare.

O objectivo deste trabalho é o de evoluir o sistema através da possibilidade da criação de grupos de utilizadores, com base em jogos de papéis onde cada jogador pode encarnar uma determinada personagem, e identificar as diferentes emoções

[2] Project Oxygen - http://www.oxygen.lcs.mit.edu/, 2004. 22 July, 2007.

expressas pelos utilizadores em cada instante, com o intuito de, o sistema ser capaz de tomar as decisões necessárias, com base nos diferentes tipos de emoções expressas pelos utilizadores.

4.1 Simulação dos utilizadores

Em cada cenário, devem existir utilizadores para que tudo isto faça sentido. Mais do que isso, os utilizadores interagem com o sistema e são, provavelmente, a parte mais imprevisível deste.

Os utilizadores podem alterar os parâmetros ambientais da casa através dos actuadores ou através das acções rotineiras que realizam em casa. Por exemplo, se o utilizador decide tomar um banho, ele está a aumentar a temperatura e humidade na casa de banho. O simples facto de interagir com certos dispositivos interfere com os parâmetros ambientais: se o utilizador liga o forno para cozinhar uma refeição, a temperatura na cozinha subirá. Isto justifica a importância de simular os diferentes utilizadores e as suas acções dentro da casa.

À plataforma VirtualECare foi acrescentada a possibilidade de adicionar vários utilizadores-tipo. Estes utilizadores são criados com base em jogos de papéis, isto é, cada utilizador-tipo vai representar um papel específico dentro do ambiente, o que o habilita a efectuar determinadas acções com base em regras previamente estabelecidas para este papel. No momento da criação destes papéis, o utilizador da plataforma de simulação deve definir as diferentes acções que cada papel poderá efectuar dentro da casa (Figura 2). A utilização dos jogos de papéis permite-nos distinguir diferentes grupos de utilizadores com características específicas.

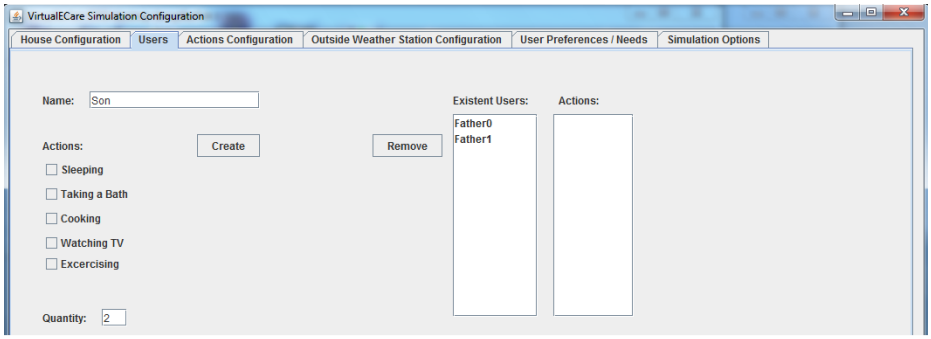


Figura 2. Configuração dos utilizadores.

Há, no entanto, dados mais importantes a serem simulados sobre os utilizadores. Como o sistema final visa monitorizar os sinais vitais dos utilizadores (Figura 3), estes também devem ser simulados para testar os mecanismos de inferência que tentam avaliar o estado de saúde dos diferentes utilizadores. A

simulação dos sinais vitais pode provocar a ocorrência de casos específicos e ver como e quão rápido o sistema reage a certas configurações de sinais vitais, podendo desta forma melhorar os mecanismos de inferência. São possíveis duas modalidades para configurar os sinais vitais dos utilizadores: *Random mode* e *Planned mode*. No *Random mode* os sinais vitais dos diferentes utilizadores são gerados de forma totalmente aleatória, enquanto que no *Planned mode* estes sinais vitais podem ser totalmente configurados.

4.2 Simulação das emoções

São fornecidas pelo VirtualECare mais algumas informações dos utilizadores que nos permitem prever ou determinar as suas emoções em cada instante. Inicialmente, estas emoções são determinadas através das preferências (temperatura,humidade e luminosidade) (Figura 3) dos diferentes utilizadores, com base em regras previamente definidas. Por exemplo, se o utilizador tem preferência por uma temperatura mais elevada e o quarto onde se encontra está frio, o utilizador pode expressar a emoção de tristeza ou desapontamento.

Para determinar as emoções é usada uma simplificação da teoria OCC [27] onde, inicialmente, apenas teremos em conta os seguintes tipos básicos de emoções: alegria, tristeza, medo, raiva, desapontamento, surpresa. Com base nas preferências dos diferentes utilizadores e nas características momentâneas de cada quarto (temperatura,humidade e luminosidade), o sistema é capaz de modelar as diferentes emoções através da conjugação das preferências do utilizador e das características do quarto em que este se encontra.

5 Conclusões e Trabalho Futuro

Como resultado deste trabalho, foi criada a possibilidade de inserção de vários utilizadores recorrendo a técnicas RPG, o que nos permite efectuar simulações (Figura 4) mais próximas da realidade dos diferentes ambientes possíveis. Ainda, foi desenvolvida uma ferramenta que permite modelar diferentes emoções, com base nas características dos utilizadores e do próprio ambiente.

Como trabalho futuro, pretende-se que o sistema utilize estas emoções para tomar as diferentes decisões possíveis, ao invés de interrogar o utilizador até que o sistema aprenda por aplicação de técnicas de CBR.

Pretende-se, ainda, no futuro, ser capaz de determinar as diferentes emoções dos utilizadores com a ajuda dos seus sinais vitais o que aumentará a aproximação à realidade já conseguida actualmente.

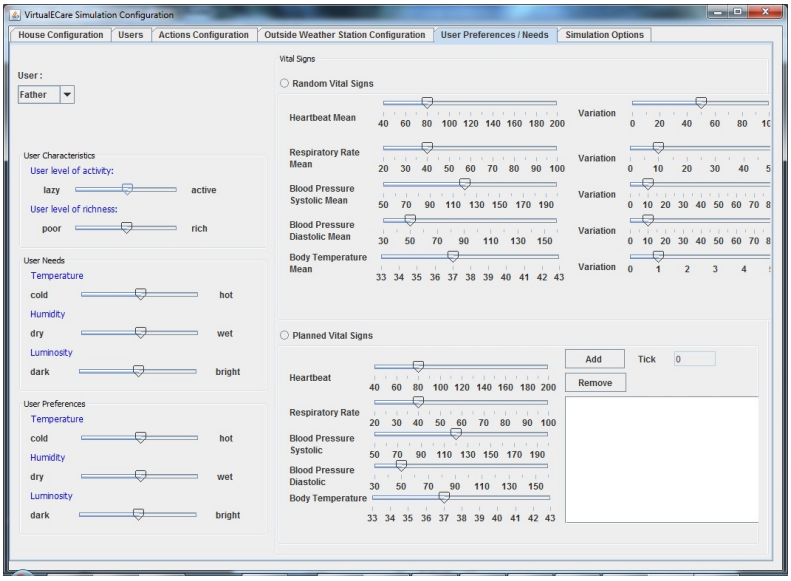


Figura 3. Configuração das preferências dos utilizadores.
[11]

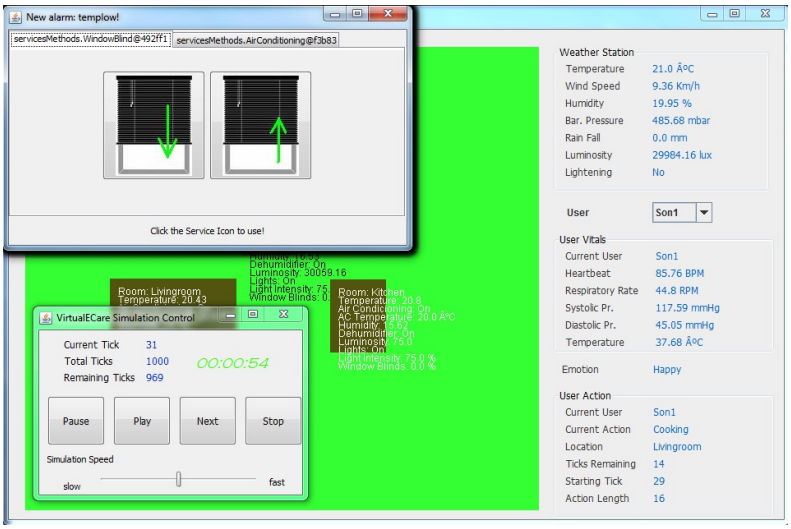


Figura 4. Simulação do ambiente.

Agradecimentos O trabalho descrito neste artigo foi parcialmente suportado pelo projecto *TIARAC - Telematics and Artificial Intelligence in Alternative Conflict Resolution Project* (PTDC/JUR/71354/2006).

Referências

1. Giovanni Acampora, Vincenzo Loia, Michele Nappi, and Stefano Ricciardi. Ambient intelligence framework for context aware adaptive applications. In *CAMP '05: Proceedings of the Seventh International Workshop on Computer Architecture for Machine Perception*, pages 327–332, Washington, DC, USA, 2005. IEEE Computer Society.
2. D. F. Adamatti. *Inserção de Jogadores Virtuais em Jogos de Papéis para Uso em Sistemas de Apoio À Decisão em Grupo: um Experimento na Gestão de Recursos Natuais*. PhD thesis, São Paulo: Escola Politécnica da Universidade de São Paulo, 2007.
3. F. Andrade. Rpg e educação - possibilidades de uso do rpg. *http://www.historias.interativas.nom.br/educ/rpgtese.htm*, 1997.
4. J.C. Augusto, P. McCullah, V. McClelland, and J.-A. Walden. Enhanced healthcare provision through assisted decision-making in a smart home environment. *2nd workshop on artificial intelligence techniques for ambient intelligence*, 2007.
5. P. Bard. On emotional expression after decortication with some remarks on certain theoretical views, parts 1 and 2. *Psychological review*, 41:309–449, 1934.
6. Olivier Barreteau, Christophe Le Page, and Patrick D'Aquino. Role-playing games, models and negotiation processes. *Journal of Artificial Societies and Social Simulation*, 6:2, 2003.
7. Magda Bercht. *COMPUTAÇÃO AFETIVA : VÍNCULOS COM A PSICOLOGIA E APLICAÇÕES NA EDUCAÇÃO*. PhD thesis, Instituto de Informática - Universidade Federal do Rio Grande do Sul - UFRGS, 2006.
8. M. Bick and T. Kummer. Ambient intelligence and ubiquitous computing. *In Handbook on Information Technologies for Education and Training*, Part I(Subpart 1):79–100, 2008.
9. Walter B. Cannon. The james-lange theory of emotions: A critical examination and an alternative theory. *The American Journal of Psychology*, 39(1/4):106–124, 1927.
10. Davide Carneiro, Paulo Novais, Ricardo Costa, and José Neves. Case-based reasoning decision making in ambient assisted living. In *IWANN '09: Proceedings of the 10th International Work-Conference on Artificial Neural Networks*, pages 788–795, Berlin, Heidelberg, 2009. Springer-Verlag.
11. Ricardo Costa, Paulo Novais, Luís Lima, Davide Carneiro, Dário Samico, João Oliveira, José Machado, and José Neves. virtualecare: intelligent assisted living. In *ehealth*, pages 138–144, 2008.
12. A. R. Damásio. O erro de descartes. emoção, razão e o cérebro humano. *São Paulo: Companhia das Letras.*, 1996.
13. Sigmund Freud. Formulations on the two principles of mental functioning. *London: Hogarth Press*, 12:213–216, 1911.
14. N.H. Frijda. The emotions. *New York: Cambridge University Press.*, 1986.
15. Kiel Mark Gilleade, Alan Dix, and Jen Allanson. Affective videogames and modes of affective gaming: Assist me, challenge me, emote me. In de Castell Suzanne and Jenson Jennifer, editors, *Changing Views: Worlds in Play: Proceedings of the 2005 Digital Games Research Association Conference*, page 7, Vancouver, June 2005. University of Vancouver.
16. M. Goreti Marreiros. *Agentes de Apoio à Argumentação e Decisão em Grupo*. PhD thesis, Escola de Engenharia - Universidade do Minho, 2007.

17. A. Herbon, A. Oehme, and E. Zentsch. Emotions in ambient intelligence-an experiment on how to measure affective states. *HCI 2006*, 2006.
18. Marta Irving, Davis Sansolo, Gustavo Melo, Ivan Burstyn, Altair Sancho, and Jean-Pierre Briot. A computer-based role-playing game for participatory management of protected areas: The simparc project. In *IV Encontro da Associação Nacional de Pesquisa e Pós-Graduação em Ambiente e Sociedade (IV ENANPPAS)*, Brasília, DF, Brasil, 6 2008. Associação Nacional de Pós-Graduação e Pesquisa em Ambiente e Sociedade (ANPPAS).
19. W. James. what is an emotion? *mind*, 9:188–205, 1884.
20. S. Klimick. *Construção de personagem & aquisição de linguagem: o desafio do rpg no ines.* PhD thesis, departamento de artes e design - puc, rio de janeiro, 2003.
21. Mit Media Laboratory and Rosalind W. Picard. What does it mean for a computer to "have"emotions? rosalind w. picard. In *In: Emotions in Humans and*, pages 115–148. MIT Press, 2001.
22. C. Lange. the emotions. reprinted in the emotions, lange and james (eds.). *new york: harner publishing co. 1967.*, 1885.
23. R.S. Lazarus. Psychological stress and the coping process. *New York: McGraw Hill.*, 1966.
24. R.S. Lazarus. Progress on a cognitive-motivational-relational theory of emotion. *American Psychologist*, 46:819–834, 1991.
25. Elisa Mattarelli, Kelly J. Fadel, and Suzanne P. Weisband. Design of a role-playing game to study the trajectories of health care workers in an operating room. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1091–1096, New York, NY, USA, 2006. ACM.
26. Marvin Minsky. Why people think computers can't. *AI Magazine*, 3(4):3–15, 1982.
27. A. Ortony, G. Clore, and A. Collins. The cognitive structure of emotions. *University Press*, 1988.
28. Costa R., Carneiro D., Novais P., Lima L., Machado J., Marques A., and Neves J. Ambient assisted living. In *Advances in Soft Computing*, volume 51, pages 86–94, Springer- Verlag, 2008. IEEE Computer Society.
29. Carlos Ramos. Ambient intelligence - a state of the art from artificial intelligence perspective. In *Portuguese Conf. Artificial Intelligence Workshops*, pages 285–295, 2007.
30. Pramila Rani and Nilanjan Sarkar. Emotion-sensitive robots - a new paradigm for human-robot interaction, 2006.
31. G. Riva. Ambient intelligence in health care. *Cyberpsychol*, 6:295–301, 2003.
32. G. Riva, F. Vatalaro, F. Davide, and M. Alcañiz, editors. *Ambient Intelligence - The evolution of technology, communication and cognition towards the future of human-computer interaction.* OCSL Press, 2005.
33. G. Schlup, Raphael P. T. de Jesus, Ricardo B. de Simas, Anita M. da Rocha Fernandes, and Rudimar L. S. Dazzi. *RPG Educacional Utilizando o Conceito de Agentes.* PhD thesis, Universidade do Vale do Itajaí, 2004.
34. M. Shuzo, J.J. Delaunay, M. Shimura, and I. Yamada. Shoji: A communication terminal for sensing and receiving ambient information. *IDETC/CIE 2009*, 2009.
35. Aaron Sloman and Monica Croucher. Why robots will rave emotions. In *IJCAI*, pages 197–202, 1981.
36. Ivanete Zuchi. *The Development of a Expert Prototype System Based on RPG techniques for the learning of Mathematics.* PhD thesis, Engenharia de Produção e Sistemas, Universidade Federal de Santa Catarina - UFSC, 2000.

O Processo ETL em Sistemas *Data Warehouse*

João Ferreira, Miguel Miranda, António Abelha e José Machado

Universidade do Minho, Departamento de Informática,
Braga, Portugal
tiago_jtx@hotmail.com
{miranda,abelha,jmac}@di.uminho.pt
http://www.di.uminho.pt

Resumo. Extração, Transformação e Carga (*Extract Transform Load* - ETL) são procedimentos de uma técnica de *Data Warehouse* (DW), que é responsável pela extracção de dados de várias fontes, a sua limpeza, optimização e inserção desses dados num DW. Este artigo tem como objectivo demonstrar o funcionamento genérico do processo ETL em sistemas DW. O processo ETL é uma das fases mais críticas na construção de um sistema DW, pois é nesta fase que grandes volumes de dados são processados. Será abordado de forma sucinta, o modo como este processamento ocorre, e ainda, as ferramentas de ETL disponíveis no mercado. Por fim, serão abordados quais os critérios a ter em consideração na escolha de uma destas ferramentas.

Palavras-chave: *Extract Transform Load* (ETL), *Data Warehouse* (DW), Ferramentas ETL .

1 Introdução

A ideia principal de um sistema de *Data Warehouse* (DW) (ilustrado na figura 1), consiste em agregar informação proveniente de uma ou mais Bases de Dados (BD), ou de outras fontes, para posteriormente a tratar, formatar e consolidar numa única estrutura de dados. Um sistema DW está associado a BD com um grande volume de dados devido quer ao volume proveniente das fontes heterogéneas quer da baixa normalização habitualmente utilizada. A estrutura de dados do DW é desenvolvida de forma a facilitar a análise desses dados. Após ser armazenada, estas informação, fica disponivel no DW ou em *DataM*arts (DM) para consultas que visam ajudar na tomada de decisão. Devido ao custo elevado, o DW muitas vezes é dividido em partes menores, nomeadamente os DM. Um DM consolida apenas as informações de uma determinada área e após a sua criação podem se unir vários DM para formarem um único DW [1].

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 757–765

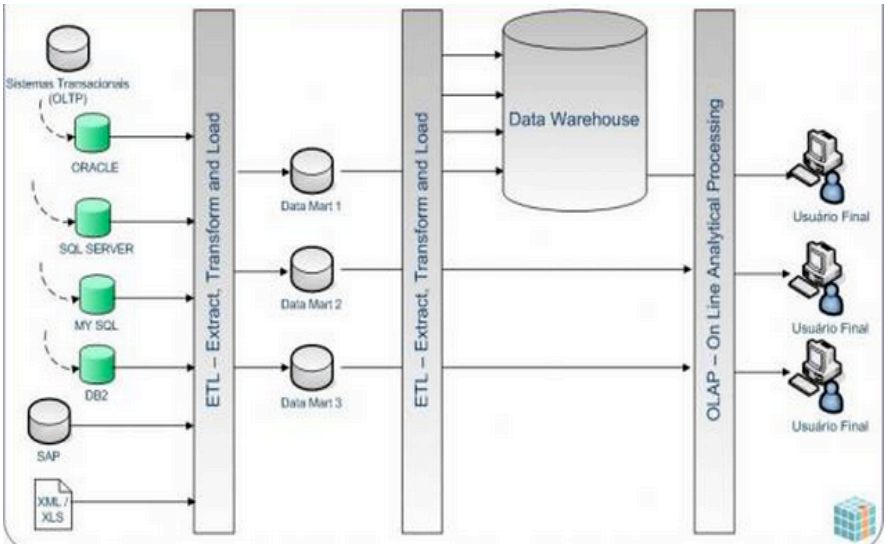


Figura 1. Esquema da Infra-estrutura de um sistema DW [1]

Para a construção de um DW são necessários diferentes passos principalmente ao nível da extracção e processamento de dados. O processo ETL destina-se à extracção e transformação dos dados e termina com a inclusão destes no DW. Esta fase caracteriza-se por englobar procedimentos de limpeza, integração e transformação de dados. Segundo a literatura este é o processo mais crítico e demorado na construção de um DW [1].

Quando o DW se encontra construído, uma das ferramentas mais utilizadas para o acesso e a análise dos dados é o *Online Analytical Processing* (OLAP). Através desta ferramenta é possível realizar o tratamento dos dados proveniente de diferentes fontes em tempo real, utilizando métodos mais rápidos e eficazes. Permite também usar uma grande variedade de ferramentas de visualizações dos dados e organizá-los através dos critérios de selecção pretendidos. A maior vantagem do OLAP é, no entanto, a capacidade de realizar análises multidimensionais dos dados, associadas a cálculos complexos, análises de tendências e modelação [3,2].

2 O Processo ETL

O ETL é um processo para extrair dados de um sistema de Bases de Dados (BD), sendo esses dados processados, modificados, e posteriormente inseridos numa outra BD. Estudos relatam que o ETL e as ferramentas de limpeza de dados consomem um terço do orçamento num projecto de DW, podendo, no que respeita ao tempo de desenvolvimento de um projecto de DW, chegar a consumir 80% desse valor. Outros estudos mencionam, ainda, que o processo de ETL tem custos na ordem dos 55% do tempo total de execução do projecto de DW [4,5,6].

A figura 2 descreve de forma geral o processo de ETL. A camada inferior representa o armazenamento dos dados que são utilizados em todo o processo. No lado esquerdo pode-se observar os dados "originais" provenientes, na maioria dos casos, de BD ou, então, de ficheiros com formatos heterogéneos, por exemplo de texto. Os dados provenientes destas fontes são obtidos (como é ilustrado na área superior esquerda da figura 2), por rotinas de extracção que fornecem informação igual ou modificada, relativamente à fonte de dados original. Posteriormente, esses dados são propagados para a *Data Staging Area* (DSA) onde são transformados e limpos antes de serem carregados para o DW. O DW é representado na parte direita da figura e tem como objectivo o armazenamento dos dados. O carregamento dos dados no DW, é realizado através das actividades de carga representadas na parte superior direita da figura.

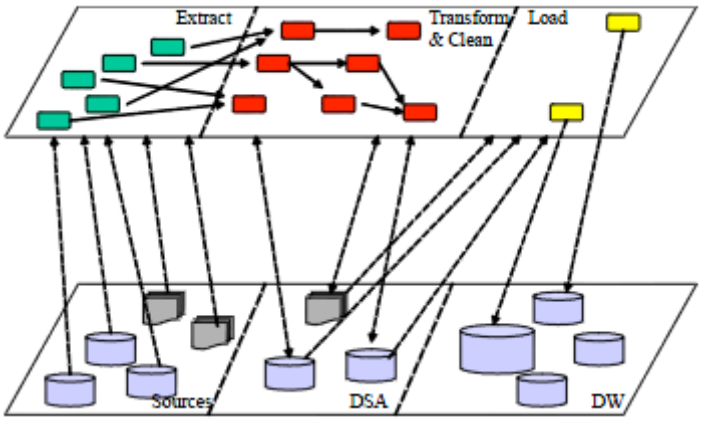


Figura 2. Ilustração do processo de ETL [13].

O ETL é um processo que se divide em três fases fulcrais:

1. Extração;
2. Transformação;
3. Carga.

Segundo alguns autores a concepção de um processo ETL incide sobre o mapeamento dos atributos dos dados de uma ou várias fontes para os atributos das tabelas do DW [7,8].

2.1 Utilização do processo ETL em BD e Ferramentas disponíveis

No DW, os dados normalmente utilizados estão localizados em BD multidimensionais. É importante que se tenha consciência que as alterações nos dados

não afectam as fontes originais, mas sim, os dados no momento de extracção para o repositório da DW. Mais ainda, que os ajustes são modelados de acordo com as necessidades do modelo de DW, atendendo assim às restrições que são necessárias para esse modelo [12].

Depois do processo de transformação ocorre o processo de carga. Neste processam-se os mapeamentos sintácticos e semânticos entre os esquemas, respeitando as restrições de integridade e criando assim uma visão concretizada e unificada das fontes. Este processo é dos mais árduos e complexos de obter devido a sua complexidade que dependerá da heterogeneidade das BD [10] [11].

No mercado existem muitas ferramentas capazes de executar processos de ETL, a tabela 1 apresenta uma visão geral da evolução destas ferramentas [3].

Tabela 1. As várias gerações de ETL ao longo dos anos

Ano	**Título**	**Significado**
Início de 1990	Codificação manual de ETL	Códigos personalizados escitos à mão
1993-1997	A primeira geração de ferramentas de ETL	Código baseado em ferramentas de ETL
1999-2001	Segunda geração de ferramentas de ETL	Código baseado em ferramentas de ETL
2003-2010	Ferramentas de ETL actualmente	A maioria das ferramentas eficientes

As ferramentas de ETL disponíveis actualmente encontram-se bem preparadas para o processo de extracção, transformação e carga. Tem-se assistido a inúmeros avanços nestas ferramentas desde 1990, estando actualmente mais direccionadas para o utilizador [3].

Uma boa ferramenta de ETL deve ser capaz de comunicar com as diversas BD e ler diferentes formatos. Actualmente a oferta é elevada, como registado na tabela 2.

Tabela 2. Diferentes ferramentas de ETL

Lista de ferramentas ETL	Versão	ETL vendedores
Oracle Warehouse Builder (OWB)	11gR1	Oracle
Data Integrator & Data Services	XI 3.0	SAP Business Objects
IBM Information Server (Datastage)	8.1	IBM
PowerCenter	9.0	Informatica
Elixir Repertoire	7.2.2	Elixir
Data Migrator	7.6	Information Builders
SQL Server Integration Services	10	Microsoft
Talend Open Studio & Integration Suite	4.0	Talend
DataFlow Manager	6.5	Pitney Bowes Business Insight
Data Integrator	9.2	Pervasive
Open Text Integration Center	7.1	Open Text
Transformation Manager	5.2.2	ETL Solutions Ltd.
Data Manager/Decision Stream	8.2	IBM (Cognos)
Clover ETL	2.9.2	Javlin
ETL4ALL	4.2	IKAN
DB2 Warehouse	9.1	IBM
Pentaho Data Integration	3.0	Pentaho
Adeptia Integration Server	4.9	Adeptia

A selecção de uma ferramenta de ETL adequada é uma decisão muito importante a ser tomada. A ferramenta de ETL opera no núcleo do DW, com a extracção de dados de múltiplas fontes e a sua transformação. Estas características tornam-na numa ferramenta acessível para os analistas de sistemas de informação.

Ao contrário de outros componentes de uma arquitectura de *Data Warehousing*, é muito difícil mudar de uma ferramenta ETL para outra, devido à falta de normas, definições de dados e regras de transformação.

Ao seleccionar uma ferramenta de ETL devem ser tomados em consideração os seguintes pontos [9]:

- Suporte à plataforma: Deve ser independente de plataforma, podendo assim correr em qualquer uma.
- Tipo de fonte independente: Deve ser capaz de ler directamente da fonte de dados, independentemente do seu tipo, saber se é uma fonte de RDBMS (*Relational Database Management System*), ficheiro simples ou um ficheiro XML.
- Apoio funcional: Deve apoiar na extracção de dados de múltiplas fontes, na limpeza de dados, e na transformação, agregação, reorganização e operações de carga.
- Facilidade de uso: Deve ser facilmente usada pelo utilizador.
- Paralelismo: Deve apoiar as operações de vários segmentos e execução de código paralelo, internamente, de modo que um determinado processo pode tirar proveito do paralelismo inerente da plataforma que está sendo executada. Também deve suportar a carga e equilíbrio entre os servidores e capacidade de lidar com grandes volumes de dados. Quando confrontados com cargas muito

elevadas de trabalho, a ferramenta deve ser capaz de distribuir tarefas entre múltiplos servidores.

- Apoio ao nível do *debugging*: Deve apoiar o tempo de execução e a limpeza da lógica de transformação. O utilizador deve ser capaz de ver os dados antes e depois da transformação.
- Programação: Deve apoiar o agendamento de tarefas ETL aproveitando, assim, melhor o tempo não necessitando de intervenção humana para completar uma tarefa particular. Deve também ter suporte para programação em linha de comandos usando programação externa.
- Implementação: Deve suportar a capacidade de agrupar os objectos ETL e implementa-los em ambiente de teste ou de produção, sem a intervenção de um administrador de ETL.
- Reutilização: Deve apoiar a reutilização da lógica de transformação para que o utilizador não precise reescrever, várias vezes, a mesma lógica de transformação outra vez.

3 Caso de estudo

Na sequência da necessidade de validar os dados dos recursos humanos de um centro hospitalar português foi extraída a informação dos seus repositórios para um ambiente de *data warehouse*. A ferramenta escolhida para o tratamento de dados e construção do repositório foi a *release* 2 da *Oracle Database* 11g, que possui embebida em si a plataforma de desenvolvimento de *data warehouse* denominada *Oracle Warehouse Builder*. A fonte principal era uma instância *Oracle 8i*, na qual estavam integrados em diferentes perfis dados de recursos humanos e outros sistemas como o de controlo de ponto.

A informação encontrava-se dispersa em mais de uma centena de tabelas com registos processados e a processar. A dispersão de informação obrigou a alterar a fundo o esquema normal de destino procurando uma normalização de nível mais baixo para a construção dos diferentes *data marts*. Desta forma foram necessários desenvolver métodos para o ETL do repositório dos recursos humanos que garantissem a qualidade da informação e permitissem a construção de um novo repositório que fosse mais adequado para *alimentar* a DW.

Nesta fase tentou-se garantir que toda a informação estava correcta e consistente, teve-se algum receio que dados incorrectos pudessem conduzir a erros críticos de tomada de decisão. Dada esta importância de detecção de erros serão de seguida explicitados alguns objectivos de teste que se estabelecem para o sistema ETL:

3.1 Preenchimento de dados

Neste teste procura-se assegurar que todos os dados esperados eram carregados.

- Comparam-se o número de registos entre os dados das fontes e o número de registos carregados para o DW.

- Comparam-se valores únicos de determinados atributos entre as fontes e os dados carregados para o DW.
- Procura-se fazer um bom esquema de dados para perceber as limitações dos valores atribuídos.
- Procura-se validar os conteúdos de cada atributo, ou seja, não permitir que por razões de codificação o limite de caracteres entre cada esquema relacional (fonte e destino) não resulta na falha do fluxo de dados.
- Transformação de Dados - Neste teste tenta-se assegurar que os dados são transformados correctamente de acordo com as regras de negócio especificadas.
- Procuram-se criar testes, os mais diversos possíveis para antever algumas situações consequentes.
- Tenta-se validar o processamento correcto de campos no ETL tais como chaves estrangeiras.
- Procura-se verificar sempre se os tipos de dados presentes no DW são os que se tinham planeado.
- E ainda procura-se testar a integridade referencial entre as tabelas.

3.2 Qualidade de dados

Neste teste procura-se assegurar que o sistema ETL rejeita ou substituí valores por defeito, corrige ou ignora dados e reporta dados inválidos.

- Procura-se realizar as conversões dos dados sempre correctamente.
- Nos casos de atributos NULL procura-se sempre inserir valores equivalentes a "desconhecido".
- Sempre que algum atributo não está correcto procura-se validar e corrigir o problema.
- Sempre que aparecem valores duplicados analisam-se os códigos e corrige-se o problema

3.3 Performance e Escalabilidade

Nesta fase procura-se, assegurar que o carregamento dos dados e a performance das interrogações são eficientes e que a arquitectura é escalonável.

- Os carregamentos de teste são efectuados com volumes de dados pequenos para garantir o bom funcionamento.
- Comparam-se estes valores de performance de carregamento do ETL para antecipar questões de escalabilidade. Assim pontos de fraqueza que sejam detectados podem ser melhorados.
- Efectuam-se operações simples com junções para validar a performance das interrogações em volumes de dados muito grandes.

3.4 Integridade de dados

Neste teste procura-se verificar que o processo de ETL funciona correctamente em relação a outros processos de *upstream* e *downstream*.

3 Conclusão

O processo ETL é o mais complexo e moroso na construção de um sistema DW, devido a aspectos já anteriormente vistos neste artigo. Nos dias de hoje são disponibilizadas diversas ferramentas de ETL no mercado, cada uma com as suas particularidades. Entre estas ferramentas destacam-se a *Oracle Warehouse Builder* (OWB), *SQL Server Integration Services*, entre outras referidas no presente artigo. As suas capacidades de tratamento e manipulação de informação, aliadas a facilidade e simplicidade de utilização, tornam-nas uma referência entre as ferramentas ETL abordadas. Na aquisição de uma ferramenta deste tipo é muito importante saber adequar essa escolha ao problema em questão, sendo que a produtividade na obtenção das informações geradas pelo DW irá reflectir o grau de acerto dessa escolha.

Referências

1.http://imasters.uol.com.br/artigo/11721/bi/arquitetura_de_data_warehouse_parte_02/imprimir acedido em 8 Junho 2010
2. Rudman, W.; Brown, C.; Hewitt, C. The use of data mining tools in identifying medication error near misses and adverse drug events. Top Health Information Management; 23(2). p. 94–103; 2002.
3. Evaluating ETL and Data Integration Platforms http://www.evolve.mb.ca/dw/etlreport.pdf acedido 8 Junho 2010
4. Cza. Shilakes, J. Tylman. Enterprise Information Portals. Enterprise Software Team, em http://www.sagemaker.com/company/downloads/eip/indepth.pdf acedido em 8 Junho 2010
5. M. Demarest, The politics of data warehousing. http://www.uncg.edu/ism/ism611/politics.pdf acedido em 8 Junho 2010
6. B. Inmon. The Data Warehouse Budget. DM Review Magazine, January 1997, em www.dmreview.com/master.cfm?NavID=55&EdID=1315
7. R. Kimbal, L. Reeves, M. Ross, W. Thornthwaite. The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying DataWarehouses. John Wiley & Sons, February 1998.
8. P. Vassiliadis. Gulliver in the land of data warehousing: practical experiences and observations of a researcher. In Proc. DMDW (Stockholm, Sweden, 2000), pp. 12.1 -12.16.
9. Rob Karel and Michael Goulde Market Overview: Open Source ETL Tools http://www.bismart.be/docs/forrester_research_market_overview_open_source_ETL.pdf acedido em 8 Junho 2010
10. Jorg, T., Dessloch, S.: Towards generating ETL processes for incremental loading. IDEAS, 101-110, 2008
11. Jorg, T., Dessloch, S.: Formalizing ETL Jobs for Incremental Loading of DataWare-houses. BTW, 327-346, 2009
12. Kimball, R., Caserta, J.: The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. John Wiley & Sons, 2004

13. Panos Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M., Skiadopoulos, S.: A generic and customizable frameworkfor the design of ETL scenarios. Information Systens 30, 492-525, 2005

Processo Clínico Electrónico Visual

Rui Marinho[1], José Machado[2], e António Abelha[2]

[1] ruipmarinho@gmail.com
[2] Departamento de Informática - Universidade do Minho - Portugal
{jmac,abelha}@di.uminho.pt

Resumo Com a crescente expansão dos sistemas de informação de saúde, o Processo Clínico Electrónico (PCE) tornou-se numa das fontes agregadoras de informação clínica mais importantes no contexto da saúde digital. Consequentemente, pede-se cada vez mais um esforço adicional aos profissionais de saúde no preenchimento de actos clínicos estruturados, tipicamente através de formulários, que se têm revelado desapropriados por serem demasiado complexos. Esta situação levou ao desenvolvimento de um novo conceito de registo de informação designada de PCE Visual, no qual o profissional de saúde regista o que vê, e não o que pretende dizer que viu. Através de tecnologia exclusivamente Web, foi possível implementar um protótipo para o registo de procedimentos, traumas e lesões em modelos anatómicos, com captação de dados estruturados com recurso a objectos gráficos, de leitura imediata, de consulta fácil e de interacção natural, preparada para suportar equipamentos sensíveis ao toque.

Palavras-Chave: processo clínico electrónico, pce, visual, svg, interactivo, gráficos, aplicação web

Abstract. With the increasing expansion of health information systems, the Electronic Health Record (EHR) has become one of the finest sources for clinical information aggregators in the context of digital health. As a consequence, health professionals are being asked to provide more thorough structured clinical statements when filling up forms, which are becoming inappropriate and overly complex. This situation led to the development of a new concept of information registration designated Visual EHR, in which the health professional registers what he sees and not what he means. Exclusively through Web technology, it was possible to implement a prototype for the registration of procedures, traumas and injuries in anatomic models, effectively capturing structured data using graphical objects, much more easier to understand and to work with, and also capable of supporting multi-touch devices.

Keywords: electronic health record, ehr, visual, svg, interactive, graphical, web application

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 767–778

1 Introdução

O Processo Clínico Electrónico (PCE) é um registo de saúde informatizado onde profissionais de saúde registam informação clínica sobre um paciente. Tem como objectivo auxiliar os sistemas de informação a reunir todos os cuidados de saúde prestados a um determinado paciente e facultar uma análise transversal do seu historial clínico em diferentes serviços e unidades médicas. Para além de informações biométricas, prescrições correntes e resultados de exames imagiológicos e laboratoriais, começam a surgir novos mecanismos mais avançados que já integram o PCE com sistemas de apoio à decisão e tarefas logísticas e contabilísticas [7].

A quantidade e a qualidade da informação disponível num PCE para os profissionais de saúde pode ter um forte impacto no seu desempenho, pois é esta que guia o seu percurso de tomada de decisão. É, por isso, fundamental que múltiplos eixos informativos se cruzem de forma relacionada e coerente.

Tecnicamente, um PCE é constituído por um conjunto de dados que se designa de *texto narrativo livre não estruturado* e por *dados codificados e estruturados*. Entenda-se por dados estruturados um conjunto agrupado de informações que do ponto de vista informático está relacionado com outro pedaço de informação. Esta ligação, descrita em termos lógicos ou a nível do modelo representa uma associação estrutural e possivelmente semântica, que permite a interpretação dos termos e dos meta-dados que são recolhidos e posteriormente processados. Isto permite que as aplicações clínicas e os agentes inteligentes associados consigam depois actuar de forma mais precisa e concertada ao nível conceptual humano, uma vez que *conhecem* o *significado* da informação com que lidam. Dificilmente se conseguem os mesmos resultados com texto narrativo livre.

Por este motivo, o uso de técnicas que permitem expandir este tipo de dados a todo o PCE tem vindo a aumentar, recorrendo-se cada vez mais a mecanismos que procuram facilitar a recolha e a análise de dados estruturados. Esta informação é muito valiosa, pois permite contribuir para a investigação clínica, para a optimização de fluxos de trabalho, para o refinamento de motores de apoio à decisão, para o melhoramento da gestão das infra-estruturas hospitalares e para o planeamento de forma mais objectiva da prestação dos serviços de saúde [9, 10, 2, 8]. Deste modo, à medida que é incentivada a captação de dados estruturados, mais rigor e objectividade são exigidos dos mesmos.

É inegável a supremacia dos dados estruturados face aos de texto livre narrativo no campo do processamento computacional. Contudo, do ponto de vista do profissional de saúde, os dados estruturados são mais complicados de gerir pois envolvem o preenchimento de variados formulários. É legítimo considerar que este cenário pode induzir uma quebra de produtividade nos profissionais de saúde, juntamente com a possibilidade acrescida de ocorrerem mais erros do que na redacção de um parágrafo de texto, devido a interfaces mal desenhadas, demasiado complexas, ou com mecanismos de validação desapropriados.

Idealmente, os dados deveriam ser capturados de forma não estruturada no decorrer da actividade médica, numa conversa entre o profissional de saúde e o paciente, e serem posteriormente processados de forma estruturada em formato

electrónico [6, 1, 11]. Até a taxa de erro deste cenário ser aceitável, novas formas de interacção Homem-Máquina terão de ser desenvolvidas, suprimindo, por um lado, as necessidades de dados estruturados e, por outro, diminuindo as barreiras de utilização.

Neste contexto, este artigo propõe o desenvolvimento de uma nova aplicação gráfica baseada apenas em tecnologia Web, recorrendo a tecnologias interactivas e de visualização para criar uma nova dinâmica na comunicação entre o profissional de saúde e o sistema de informação do PCE, de modo a auxiliá-lo a registar visualmente informação médica, sem recurso a formulários ou outros mecanismos de complexidade equivalente. Esta aplicação Web permitirá aos profissionais de saúde recorrer a objectos gráficos para efectuarem uma leitura rápida da informação clínica disponível, registar procedimentos, lesões e traumas através de ferramentas gráficas clínicas pré-definidas, navegar entre níveis de detalhes anatómicos de acordo com o serviço onde se encontram, e colaborar durante um acto médico. É esperado que esta aplicação Web contribua para um aumento da estruturação dos dados e, simultaneamente, reduza a possibilidade de introdução de erros, aumente a produtividade dos profissionais de saúde e proporcione informação estruturada de qualidade.

Este artigo está organizado em mais três secções. A Secção 2 descreve o trabalho relacionado com visualização de dados clínicos; a Secção 3 apresenta os requisitos necessários para desenvolver a aplicação Web proposta, bem como a sua arquitectura e implementação. Na Secção 4 são apresentadas as conclusões finais e são apresentados alguns exemplos de melhorias possíveis no futuro.

2 Trabalho Relacionado

2.1 Agência de Interoperação, Difusão e Arquivo

Visão Geral A única implementação conhecida de uma interface Web de registo clínico electrónico visual é no *Quadro de Registo de Procedimentos (QRP)*, integrado na *Agência de Interoperação, Difusão e Arquivo (AIDA)*. É uma plataforma que resulta de parcerias de investigação entre a Universidade do Minho e unidades de saúde Portuguesas e tem como objectivo promover o arquivo e a difusão dos Meios Complementares de Diagnóstico (MCDTs) e a terapêutica ao nível da unidade hospitalar, centro hospitalar ou unidade local de saúde, bem como a consolidação electrónica às escalas regional e nacional [3].

O QRP, inserido no sistema de PCE da AIDA [4], é uma área de trabalho clínica pioneira que possibilita aos profissionais de saúde registarem procedimentos através de ferramentas gráficas no PCE de cada paciente. O QRP é composto por duas acções principais: o índice de registos e vista de procedimentos. A primeira contém um mapa de visualização dos registos efectuados até ao momento no paciente, sendo utilizado uma representação 3D de um modelo anatómico que varia apenas com o sexo do paciente. Estes registos estão indicados através de um círculo colocado na imagem anatómica directamente no local onde o procedimento ocorreu. Também é incluído um resumo textual com o nome do procedimento e a

data de colocação de todos os registos efectuados, estando organizado por zonas do corpo humano pré-definidas. Os registos podem ser retirados, alterando-se a sua cor no mapa de visualização, e serem acompanhados de observações clínicas. No acto de registo, quando uma zona do corpo é seleccionada, são apresentados todos os procedimentos disponíveis nessa área, juntamente com parâmetros complementares, caso existam.

Limitações As principais limitações desta ferramenta estão relacionadas com a biblioteca de imagens anatómicas disponível e com a possibilidade de se registarem apenas procedimentos.

A plataforma permite apenas carregar três modelos (Homem adulto, Mulher adulta e Criança), limitando o registo de procedimentos a apenas uma área muito abrangente. A única perspectiva existente impede que sejam registados pormenores em locais mais minuciosos como, por exemplo, na retina de um paciente, pois o círculo correspondente a este procedimento ocuparia toda a área do olho representado.

Por outro lado, a possibilidade única de registar procedimentos, através de um marcador circular de tamanho fixo, deixa de fora outro tipo de registos como lesões e traumas, de igual modo importantes no contexto do diagnóstico clínico. Esta característica não permite, por exemplo, que sejam demarcadas áreas com polígonos irregulares, indicadores de feridas actuais ou lesões na pele.

2.2 Soluções Comerciais

Visão Geral Apenas uma solução comercial é conhecida, embora ainda em fase de testes, e foi criada no Laboratório de Investigação da IBM Zurique. Trata-se de um sistema que permite a consulta de registos médicos num ambiente 3D através de uma aplicação de Desktop. Recorrendo aos modelos anatómicos do corpo humano, alinhados de uma forma semelhante à da navegação geoespacial como no Google Earth, este sistema foi apelidado pela IBM como Motor de Mapeamento Simbólico e Anatómico. Os registos são mostrados através de setas posicionadas num eixo tridimensional na representação do corpo humano, indicando as áreas em que está disponível informação clínica. Ao seleccionarem qualquer uma destas setas, os profissionais de saúde conseguem obter todo o tipo de informação associada a essa área - registos textuais, resultados laboratoriais e imagens de MCDTs. Também é possível efectuar pesquisas semânticas pois é utilizada a nomenclatura médica sistematizada SNOMED para criar uma ponte entre os conceitos gráficos e os documentos de texto não estruturados. Outras funcionalidades incluem a possibilidade de inspecção de órgãos e dos sistemas circulatório, muscular e nervoso, bem como novos mecanismos que procuram dotar a aplicação de inteligência artificial. [5].

Limitações A plataforma da IBM integra unicamente dados de sistemas heterogéneos para os apresentar no sistema visual de forma agrupada e contextualizada. Contudo, a introdução de dados no sistema continua a depender dos

habituais processos de registar dados clínicos, algo que não foi abordado por esta solução. Tal como referido anteriormente, é essencial que as barreiras de entrada nos sistemas digitais sejam diminuídas, começando principalmente pela base fundamental do PCE.

É também de realçar que esta aplicação foi desenvolvida para ambiente de Desktop. Mesmo que o recurso à tecnologia 3D possa ter estado na origem desta opção, também esta pode ser utilizada em ambientes Web (através de WebGL). Assim, esta solução não tira partido da ubiquidade da Web e das potencialidades colaborativas que esta oferece, dificultando também o acesso multi-plataforma.

2.3 Outras Referências

Este conceito de PCE visual ainda se encontra, aparentemente, em fase embrionária. A actividade científica nesta área é muito residual e, quando existe, geralmente remete para a integração de MCDTs no PCE através de sistemas de Comunicação e Arquivamento de Imagens melhorados. Na área das aplicações Web, a potencialidade das tecnologias de visualização também parece ter maior impacto em Sistemas de Informação Geográfica, onde a sua utilização é predominante.

3 Solução Proposta

3.1 Introdução

Nos últimos anos tem-se vindo a atravessar uma importante evolução na convergência entre aplicações de Desktop e a Web, originando software conhecido como *Rich Internet Applications*. Duas das mais importantes características destas aplicações resumem-se a colaboração e interacção. Por colaboração quer-se mencionar os aspectos sociais que permitem a colaboração entre pessoas e a partilha de serviços e dados através da Web. Contudo, outro aspecto de igual importância é a interacção. As novas tecnologias tornam possível a construção de aplicações Web que se comportam de forma muito semelhante às aplicações de Desktop, permitindo, por exemplo, a actualização de um elemento de interface sem ser necessário recarregar toda a página de cada vez que algo muda. Estas aplicações Web combinam princípios de interface e paradigmas de usabilidade, em conjunto com tecnologia robusta, para transmitir a sensação de que se está a executar uma aplicação de Desktop.

Apesar dos progressos na mais recente geração de *browsers* e na introdução de novos standards como o HTML5, as soluções actuais para sistemas de informação na Saúde ainda não tiram total partido das potencialidades da Web. Parca em contextualização semântica e difícil de inserir, a informação no PCE permanece parcialmente estruturada e difícil de interpretar. Os novos avanços na tecnologia Web estão a proporcionar oportunidades incríveis na evolução da qualidade dos cuidados de saúde prestados e no desenvolvimento de interfaces inovadoras que agora se expandem para outra área - a da visualização.

A contextualização gráfica não permitirá apenas aumentar a qualidade do diagnóstico por parte do profissional de saúde, mas também contribuirá para melhorar a comunicação com o paciente, que poderá compreender melhor aquilo que o afecta. Ao entender de forma clara a situação, o paciente também poderá ter uma resposta mais adequada ao diagnóstico ou à terapêutica.

Estes conceitos serviram como base para o desenvolvimento das soluções já analisadas, mas ainda foram pouco explorados. É necessário continuar a apostar no desenvolvimento de novas interfaces Homem-Máquina que procurem ultrapassar as limitações destas soluções e potencializem as tecnologias existentes. Foi com este intuito que se desenvolveu o *Rokee*, nome de código para caso de estudo que se apresenta de seguida.

3.2 Análise de Requisitos

O QRP da AIDA-PCE prova que a Web já fornece a tecnologia necessária para a construção de interfaces que melhorem a interacção entre profissionais de saúde e as aplicações de foro clínico, contribuindo para um aumento da qualidade da informação registada. Com o *Rokee* procurou dar-se continuidade a este progresso, mas sobretudo inovar em determinados aspectos para que estes profissionais se possam concentrar mais na prática clínica e menos na tecnologia que os rodeia. Para tal, o seguinte conjunto de requisitos foi proposto para que o *Rokee* acrescentasse valor às outras soluções já estudas, considerando sempre que o ambiente de desenvolvimento escolhido é a Web:

- Aceder rapidamente às principais informações relacionados com o paciente, como dados pessoais, do processo e do Serviço em que se encontra internado.
- Navegar por data entre registos, recorrendo a lógica que permita gerir a migração de dados do *dia anterior* para o *dia actual*.
- Aceder a quadro visual com capacidade para registar procedimentos, lesões ou traumas, de acordo com a necessidade do acto clínico.
- Navegar em imagens de detalhe, agrupadas por categorias e contextualizadas com o Serviço em que o paciente se encontre internado.
- Seleccionar ferramentas que variem de acordo com o tipo de registo pretendido.
- Enumerar todos os registos efectuados na imagem de trabalho principal e em imagens secundárias, tendo estas maior nível de detalhe.
- Associar visualmente cada um dos registos gráficos à sua descrição textual (legendagem).
- Submeter observações de acordo com o registo seleccionado na imagem de trabalho e consultar o seu histórico.
- Possibilitar a retirada de registos de uma imagem com uma observação associada, caso seja pretendido.

3.3 Arquitectura do Sistema

O *Rokee* foi desenvolvido em PHP, com base no paradigma Modelo-Apresentação-Controlador da *Symfony Framework*, e implementa uma arquitectura de comu-

nicação Cliente-Servidor, como demonstrado na Figura 1. A interface é apresentada ao utilizador da primeira vez que é carregada (1) e sempre que é detectado um evento (p.e. carregar no botão de selecção do tipo de registo) dispara-se um pedido *XMLHttpRequest* (2) através de *Java Script* (*jQuery*) ao servidor (3). Este é processado de acordo com a lógica em prática e são devolvidos os respectivos dados da resposta (4). Geralmente este tipo de lógica implica um acesso ao SGBD onde são gravados os dados, embora esta interacção não seja obrigatória. Uma função de *callback* do pedido *XMLHttpRequest* trata de analisar esses dados e de determinar o que fazer com eles (5). Na Figura 1 é indicado um conjunto de dados muito frequente neste tipo de resposta (HTML e CSS), pois pode ser directamente injectado no DOM (HTML). Contudo, os dados podem ser recebidos em XML e JSON, entre outros formatos, e depois processados da forma mais adequada.

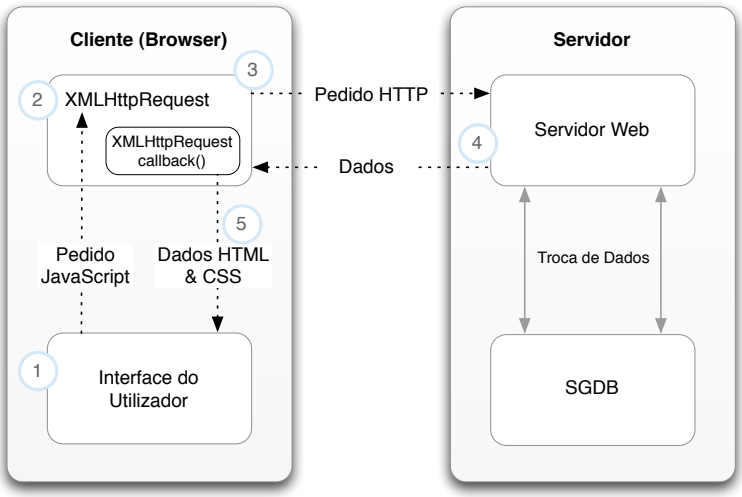


Figura 1. Arquitectura Cliente-Servidor em funcionamento no *Rokee*.

Quando se pretende desenvolver uma aplicação Web com interacções complexas e uma experiência rica para o utilizador numa vasta gama de *browsers*, a tecnologia *Flash* da Adobe é frequentemente a escolhida. Contudo, quando se define como prioritário o acesso multi-plataforma, a acessibilidade e a ubiquidade numa aplicação Web, a única solução possível é a utilização de standards Web abertos que não exijam a instalação de software de terceiros. Por este motivo, entre as diferentes tecnologias de visualização disponíveis na Web, a única que satisfaz todos estes requisitos é o formato *Scalable Vector Graphics* (SVG), que possibilita a visualização de gráficos vectoriais e animações em XML.

3.4 Funcionamento

A interface deste módulo está dividida numa área de registos e noutra de observações. Na primeira encontram-se as ferramentas de carácter genérico, como a selecção de objectos, o desenho de linhas e a inserção de texto, e as clínicas, que são utilizadas para adicionar registos, e que variam de acordo com o tipo de registo seleccionado na área de trabalho (lesões, procedimentos ou traumas). Uma área composta por três separadores, *Processo*, *Registos* e *Imagens*, mostra a informação clínica relevante em cada um destes contextos. Na segunda área os profissionais de saúde podem acrescentar e consultar observações aos registos clínicos efectuados, bem como proceder à retirada destes últimos.

Quando se entra neste modo, todo a informação do paciente é recuperada da base de dados para preencher o separador *Processo*. Do seu perfil obtêm-se os dados biométricos e do processo os dados relativos ao serviço onde se encontra internado e ao episódio presente.

A área de trabalho é composta uma imagem de fundo que é automaticamente carregada quando se inicializa este módulo. Esta imagem é recuperada da base de dados de acordo com o serviço onde o paciente se encontra. Sobre esta imagem existe um quadro de desenho invisível no qual são registados os actos clínicos no formato SVG, automaticamente gravados em base de dados, de acordo com a ferramenta de desenho correspondente.

Cada uma das ferramentas pode ter parâmetros associados que permitem acrescentar informação complementar relacionada com o acto clínico. *Calibre*, *Vias*, *Joules* e *Localização* são exemplos de parâmetros disponíveis. O seu preenchimento é opcional e pode ser efectuado no acto de colocação de um registo. Podem ser posteriormente consultados e alterados, sendo apenas necessário seleccionar o registo clínico na imagem de trabalho no qual se deseja executar esta acção.

Há um mapeamento interno em *Javascript* que depois relaciona uma ferramenta clínica com um objecto gráfico. Por exemplo, um *Cateter Venoso Central* (procedimento) produzirá sempre um círculo de tamanho variável e cor amarela, enquanto que uma *ferida actual* (lesão) permitirá construir um polígono irregular. De forma a facilitar o reconhecimento do tipo de registo em causa, todos os procedimentos encontram-se mapeados a círculos de diferentes cores, conforme a categoria em que se encontram, e as lesões e traumas a polígonos, também com cores distintas..

O reconhecimento programático das acções que o profissional de saúde está a executar na plataforma (eventos) permite construir uma interface que responda naturalmente às suas acções, de forma imediata e não intrusiva.

Registos A componente de listagem no separador *Registos* é constituída por uma representação textual de todos os registos disponíveis num determinado contexto. Encontra-se dividido em quatro painéis distintos: registos na imagem actual (a que está a ser mostrada na área de trabalho), em imagens pertencentes à mesma categoria, noutras categorias e em imagens, serviços e/ou episódios diferentes.

Cada painel contém um título com indicação sobre a categoria a que esses registos pertencem (p.e. *Imagem Actual*) e o número total de registos existentes nesse mesmo tipo. Adicionalmente, na área que serve de contentor aos registos textuais, a ordem que foram adicionados, o título da ferramenta utilizada e a data da sua colocação. Caso o registo tenha sido retirado, também é apresentada a data de remoção a vermelho. Todos os parâmetros também são listados se tiverem sido preenchidos.

Para terminar, a listagem de registos suporta ainda um mecanismo de *overflow* que permite a introdução contínua de novos registos textuais sem aumentar a área ocupada pelo contentor principal. Em comparação com as habituações barras de navegação, este sistema contempla eventos especiais de arrastar e largar, promovendo a adaptação a dispositivos multi-toque e facilitando o acesso a conteúdos extensos.

Esta área é actualizada sempre que o contexto da imagem actual é alterado mediante o uso de pedidos assíncronos.

Imagens Já foi referido o comportamento da imagem na área de trabalho quando o módulo de edição é inicializado. Contudo, uma das grandes vantagens do *Rokee* face às limitações das outras soluções estudadas é permitir o registo de informação clínica em imagens de detalhe. Esta funcionalidade só é viável se a estrutura de imagens associada a esse serviço for correctamente configurada através da interface administrativa, como detalhado mais adiante. Não obstante, o separador *Imagens* possibilita a navegação em níveis de detalhe, podendo-se ir diminuindo sucessivamente a área abrangida, desde que haja suporte imagiológico correspondente na biblioteca anatómica carregada no sistema.

Por exemplo, considerando o serviço de *Oftalmologia* (nível 0), é possível ter uma visão global da face quando se inicia o módulo de edição, sendo esta a imagem de trabalho por omissão. No entanto, no separador *Imagens* são mostradas categorias anatómicas dentro de *Oftalmologia*, como *Olho* (nível 1), *Retina* (nível 2) e *Mácula* (nível 2). Se houver necessidade de registar dados clínicos na *Retina*, pode-se simplesmente seleccionar uma das imagens disponíveis nessa sub-categoria.

A ideia é permitir um constante aumento do nível de detalhe à medida que se vai caminhando na árvore anatómica associada a um serviço. Assim promove-se o registo rigoroso e de qualidade de informação clínica, por intermédio de uma interface simples de usar.

Sempre que uma categoria de imagens é seleccionada são automaticamente obtidas as imagens associadas a essa categoria, bem como as que pertençam a sub-categorias de primeiro nível (filhos). Desta forma o profissional de saúde poderá ter sempre a noção se pretende saltar para o nível de detalhe seguinte ou se está satisfeito com o detalhe apresentado pela actual categoria.

Observações Nas mais variadas situações, um profissional de saúde tem necessidade de complementar o registo de um acto clínico com observações. Pode anexar, por isso, notas a um registo clínico, mediante uma interface que envolve

apenas a introdução do conteúdo da mensagem. A área do histórico de observações adjacente é automaticamente actualizada após o correcto envio da nota clínica e sempre que se selecciona um registo na área de trabalho.

Administração Foi também implementada uma interface administrativa para gerir a biblioteca anatómica em utilização no sistema. Os mecanismos implementados no âmbito da gestão de imagens permitem que esta plataforma seja utilizada em vários Serviços na mesma unidade de saúde, com apenas uma base de instalação. Para tal é necessário que o modelo das categorias de imagens suporte múltiplas árvores aninhadas, em que cada uma delas corresponde a um Serviço da unidade de saúde.

Dentro de cada categoria (ou Serviço), é possível ir construindo uma árvore com sub-categorias de estruturas anatómicas, de acordo com a organização do grupo clínico de cada um desses serviços. A grande vantagem deste sistema é que não requer que seja seguida de forma rígida uma estrutura anatómica pré-definida, que pode não satisfazer todos os profissionais de saúde. Todas as estruturas são viáveis e aceites pela plataforma.

A interface administrativa que dá corpo a estes mecanismos foi desenvolvida com apenas um propósito - o de organizar imagens de forma idêntica a um gestor de ficheiros num sistema operativo, através da técnica de *drag-n-drop*. A primeira acção consiste na activação do serviço na unidade hospitalar, para que possa ser criada uma raiz dedicada a este serviço. De seguida é necessário gerir a árvore de categorias e sub-categorias dentro desse serviço, isto é, a estrutura anatómica pretendida pela equipa de profissionais de saúde. Cada categoria pode ser renomeada ou apagada caso tenham havido enganos, e arrastada e largada entre diferentes posições na árvore.

Após a estrutura estar concluída chega o momento de gerir as imagens anatómicas associadas a esta. Depois de se entrar no serviço pretendido, há três áreas de merecida importância: a navegação na árvore, a área da estrutura seleccionada e a área da galeria. Através da navegação na árvore é possível ir associando imagens, bastando arrastar imagens da área da galeria para a área da estrutura escolhida. O separador *Imagens* tira de imediato partido destas alterações, sem ser necessária mais nenhuma intervenção por parte do profissional de saúde.

4 Conclusão e Trabalhos Futuros

Neste artigo foi abordada a problemática da introdução de dados estruturados no âmbito do PCE e apresentada uma solução possível para diminuir a complexidade que envolve a captação deste tipo de informação. O protótipo daí resultante - *Rokee* - provou ser uma ferramenta capaz de responder a este desafio, recorrendo unicamente a tecnologia Web aberta, ubíqua e multi-plataforma. A utilização de gráficos vectoriais nativos na Web garante o seu correcto funcionamento em qualquer dispositivo com acesso a um browser, independentemente de ser um Desktop ou um equipamento móvel. Esta plataforma de trabalho, ainda

numa fase de protótipo, abre portas à expansão de muitas outras áreas do conhecimento clínico à era das interfaces ditas *naturais*. Será interessante testar este protótipo num ambiente clínico real, com uma biblioteca de imagens adequada, particularmente em dois cenários: como evolução da solução QRP analisada no âmbito da AIDA e como uma nova plataforma noutro sistema diferente.

Espera-se que, no futuro, o *Rokee* venha a tirar ainda mais partido das novas tecnologias do standard HTML5, efectuando *caching* local de imagens da biblioteca anatómica, apresentando em tempo real registos colocados por outros profissionais de saúde e contextualizando a interface conforme a localização física do paciente. Ao nível das interfaces multi-toque, também se pretende expandir o suporte a todo o fluxo de trabalho para que funcione de forma transparente na nova geração de equipamentos móveis (como *iOS*, *Android* e *Windows Mobile*). O suporte para gráficos 3D na Web começa também a ganhar forma, de modo que poderão ser implementados mecanismos de visualização que tirem partido da modelação a 3D, facilitando ainda mais o registo de informação clínica de detalhe.

Para terminar, a longo prazo é certo que haverá uma convergência entre todos os sistemas de uma unidade de saúde. A representação de modelos anatómicos a duas ou três dimensões é um enorme passo face às ilustrações em caneta e papel, mas o futuro permitirá integrar de forma transparente imagens resultantes de meios imagiológicos (como ressonâncias magnéticas, por exemplo) directamente nesta plataforma. Não serão então representações anatómicas, mas sim a verdadeira anatomia de um paciente.

Referências

[1] J. S. Ash et al. «Types of unintended consequences related to computerized provider order entry». In: *Journal of the American Medical Informatics Association* 4.13 (2006), pp. 547–556.

[2] J. Brender, C. Nohr e P. McNair. «Research needs and priorities in health informatics». In: *International Journal of Medical Informatics* III.4 (2000), pp. 257–289.

[3] Centro de Competência em Informática Médica do Departamento de Informática da Universidade do Minho. *Suite de Produtos AIDA: Poster AIDA*. 2009. URL: `http://gia1.di.uminho.pt/aida/poster_aida_files/slide0003.htm` (acesso em 13/06/2010).

[4] Centro de Competência em Informática Médica do Departamento de Informática da Universidade do Minho. *Suite de Produtos AIDA: Poster AIDA-PCE*. 2009. URL: `http://gia1.di.uminho.pt/aida/poster_pce_files/slide0003.htm` (acesso em 13/06/2010).

[5] Robert N. Charette. *Visualizing Electronic Health Records With "Google-Earth for the Body"*. 2009. URL: `http://spectrum.ieee.org/biomedical/diagnostics/visualizing-electronic-health-records-with-googleearth-for-the-body` (acesso em 14/06/2010).

[6] R. H. Dykstra et al. «The extent and importance of unintended consequences related to computerized provider order entry». In: *Journal of the American Medical Informatics Association* 4.13 (2007), pp. 415–423.

[7] P. J. Edwards et al. «Evaluating usability of a commercial electronic health record: a case study». In: *International Journal Human-Computer Studies* 66 (2008), pp. 718–728.

[8] European Commission. *Connected Health: Quality and Safety for European Citizens*. 2006. URL: `http://ec.europa.eu/information_society/newsroom/cf/itemdetail.cfm?item_id=2788` (acesso em 13/06/2010).

[9] A.M. van Ginneken. «The computerized patient record: balancing effort and benefit». In: *International Journal of Medical Informatics* II.1 (2002), pp. 97–119.

[10] J. Grimson. «Delivering the electronic healthcare record for the 21st century». In: *International Journal of Medical Informatics* II.64 (2001), pp. 111–127.

[11] P. Hartzband e J. Groopman. «Off the record: avoiding the pitfalls of going electronic». In: *The New England Journal of Medicine* 16.358 (2008), pp. 1656–1658.

Sistema de Resolução Online de Conflito para Partilhas de bens – Divórcios e Heranças

Ana Café[1], Davide Carneiro[2], Paulo Novais[2] and Francisco Andrade[3]

[1] Universidade Católica de Angola, Faculdade de Engenharia Informática, Luanda, Angola
aclaudia.cafe@gmail.com
[2] Departamento de Informática, Universidade do Minho, Braga, Portugal
{dcarneiro, pjon}@di.uminho.pt
[3] Escola de Direito, Universidade do Minho, Braga, Portugal
fandrade@direito.uminho.pt

Abstract. Em diversos sectores da sociedade, a resolução de litígios pelos tribunais tem se revelado menos viável, mais morosa e custosa. Para contornar algumas das imperfeições dos sistemas jurídicos convencionais surgiram os processos de resolução alternativas de conflito (ADR). Devido aos avanços tecnológicos, o surgimento da Internet e com isso também novas formas de conflitos, a ADR teve necessidade de adaptar e melhorar os seus processos a fim de dar respostas às mudanças provocadas. Assim, sistemas capazes de suportar diferentes abordagens da ADR foram criados passando a denominar-se sistemas de resolução online de conflitos (ODR). A negociação assistida é uma das abordagens da ODR e é utilizada em várias situações de conflitos. O sistema UMCourt Partilha tem na base esta abordagem e foi desenvolvido para auxiliar situações de partilhas de bens em caso de divórcios e heranças contemplando conceitos da lei, técnicas de inteligência artificial e teorias de jogo.

1 Introdução

Cada pessoa reage de maneira diferente diante de um conflito. Durante muito tempo, a única solução considerada era levar o caso a tribunal. Assim, existe um demandante (pessoa que se sente lesada) que apresenta queixa ao tribunal contra um réu dando início ao processo da litigação. As partes apresentam as suas circunstâncias factuais a um juiz ou júri que deverá decidir a sentença para o caso. No entanto, a litigação deveria ser considerada como último recurso para a resolução de conflitos tendo em conta as suas características menos favoráveis. A Nationwide Academy for Dispute Resolution (UK) mencionou, a provável experiencia intimidante para as partes, o dispêndio do tempo com os encontros entre as partes e seus advogados, o quão afectada uma relação pode ficar – principalmente pelo seu aspecto ganha/perde, o quão caro este género de processo pode ser – tendo em conta os custos judicias e os honorários, como sendo algumas delas. Nos finais do século XX, processos mais amigáveis de resolução de conflitos passaram a ser considerados – os chamados

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 779–790

processos de Resolução Alternativa de Conflitos (ADR[2]). A ADR proporciona "alternativa" aos tribunais para a resolução de conflitos com processos que não fazem parte dos sistemas judiciais governamentais. No entanto, segundo [1] houve uma evolução no conceito da ADR. Ela deixa de ser simplesmente uma técnica para resolução de conflitos sem a litigação passando a ser uma técnica apropriada no contexto de resolução de conflitos no geral. Com esta mudança, a litigação poderia passar a ser considerada como um dos muitos processos de resolução de conflitos. Os principais processos da ADR são a mediação, a negociação e a arbitragem.

Posteriormente, a criação da *World Wide Web* (WWW) causou impacto na sociedade provocando mudanças de comportamentos quanto à execução de tarefas em diversos sectores. Especificamente, as maneiras de celebrar contractos progrediram até ao ponto em que a presença física deixou de ser essencial sendo compensada por meios tecnológicos. Destes novos comportamentos também advieram novos géneros de conflitos. Portanto, a ADR precisou de suporte tecnológico para suas abordagens legais, tendo também em conta os novos casos que surgiram com a evolução tecnológica. Para dar respostas a estas abordagens, têm vindo a ser desenvolvidos sistemas focados em ajudar as partes a resolver os seus conflitos através de novos meios tecnológicos. Estes sistemas de suporte à ADR são denominados sistemas de Resolução Online de Conflitos (ODR[3]). Neste sentido, a Inteligência Artificial (AI[4]) revela-se bastante interessante a explorar quando associada à ODR, por almejar a criação de sistemas capazes de optimizar processos complexos para resolução de conflitos, o que requer consideráveis engenhos e perícia em vários assuntos.

Neste artigo apresentamos uma visão geral da ODR e suas vantagens. De seguida apresentamos o algoritmo Adjusted Winner, algoritmo este que serve de base para o algoritmo de partilha apresentado neste protótipo assim como a nossa adaptação e definição para uso do Raciocínio baseado em casos.

2 Resolução de Conflitos em linha

A ODR tem sido vista como a abordagem da ADR que se apoia nos meios tecnológicos para facilitar a resolução de conflitos, ou ainda, considerando a componente "on-line", é vista como um ambiente virtual no qual as partes possam reunir-se para resolver suas diferenças. Porém, a ODR foi além de permitir o simples suporte aos processos da ADR. Não se restringiu ao suporte da arbitragem, negociação e mediação convencionais, mas também explorou processos além do alvo da ADR (nomeadamente, a negociação automatizada ou *blind-bidding*[5])[2]. A visão usual da ODR como sendo o equivalente tecnológico da ADR também tem mudado com a criação de novos processos baseados em ambientes Web.

Segundo [8], a razão pela qual as pessoas optam pelos métodos alternativos de resolução de conflitos, é a possibilidade de obter melhores resultados dos que se pudessem obter sem eles. A definição dos limites aceitáveis num processo, ajuda as

[2] Do Inglês Alternative Dispute Resolution
[3] Do Inglês Online Dispute Resolution
[4] Do Inglês Artificial Intelligence
[5] Do Inglês licitação cega

partes a definir suas prioridades e interesses básicos. De acordo com esta linha de pensamento podemos dizer que para um melhor suporte à resolução de conflitos, é importante que este género de sistema forneça informações sob duas perspectivas: (1) Ajude a preservar a parte de aceitar um acordo que deveria rejeitar; (2) Ajude a explorar as vantagens para chegar a um acordo que melhor o favorece. Sendo assim, principalmente nas negociações baseadas em interesses, as partes precisam saber qual a sua BATNA – Melhor Alternativa para um Acordo Negociado (Best Alternative to a Negotiated Agreement) e WATNA – Pior Alternativa para um Acordo Negociado (Worst Alternative to a Negotiated Agreement) [13]. Isso ajudará as partes a ter melhor percepção sobre os possíveis acordos que surgirem, no que toca aceitá-los ou recusá-los. Isto é, por um lado tomando conhecimento de sua BATNA, cada parte fica "mais protegida contra acordos que devam ser rejeitados" estando em melhores condições para "alcançar um acordo que melhor satisfaz seus interesses" [14]. Por outro lado, conhecer a sua WATNA permite às partes ter noção dos piores resultados que poderiam advir de um confronto judicial [13], reduzindo expectativas excessivamente optimistas.

Existem hoje diferentes géneros de sistemas de ODR. [4] classifica os sistemas de ODR em duas categorias: primeira e segunda geração. A primeira geração é caracterizada por sistemas sem autonomia quanto à resolução dos processos. O homem continua a ter um papel importante neste género de sistema onde a tecnologia actua apenas como uma ferramenta de suporte à decisão, estabelecendo a comunicação entre as partes ou automatizando tarefas simples [3]. Por exemplo, o CyberSettle[6] é um sistema de resolução de conflito automatizado que permite às partes resolver os seus conflitos de maneira rápida e confidencial.

A segunda geração é a expectativa dos novos sistemas ODR que terão como meta a resolução de conflitos de forma autónoma. Estes sistemas deixam de ser meras ferramentas e passam a fazer análise de casos e definição de estratégias e soluções. O objectivo é o de reduzir a intervenção humana na resolução de conflitos [4]. Brevemente estes sistemas poderão actuar como agentes autónomos. Claramente este género de sistema necessita de uma componente "inteligente" e conhecedora das áreas de conflitos para atingir este requisito. Assim, a inteligência artificial é uma das áreas de conhecimento que tem sido explorada e já são visíveis resultados neste sentido. Tomando como exemplo a área do direito de família, é possível identificar os seguintes sistemas: Family_Winner [10] que utiliza teoria de jogos e heurísticas; Expertius [5] que combina inteligência artificial e leis; e Smartsettle [3] que é um sistema de negociação online.

3 UMCourt

UMCourt é uma plataforma para ODR que está a ser desenvolvida na Universidade do Minho no contexto do projecto TIARAC (*Telematics and Artificial Intelligence in Alternative Conflict Resolution*). O principal objectivo do projecto é o de analisar o

[6] CyberSettle blind bidding system é promovido e comercialmente disponível em www.cybersettle.com. O Cybersettle foi integrado no estado de arte dos sistemas disponíveis na web em 1998.

papel que as técnicas da Inteligência Artificial, mais particularmente as técnicas baseadas em agentes, podem ter no domínio da ODR com o objectivo de tornar o processo mais rápido, simples e proveitoso para as partes. Assim, UMCourt resulta numa arquitectura na qual serviços orientados ao ODR podem ser implementados, usando como suporte as ferramentas desenvolvidas no âmbito do projecto. Estão a ser desenvolvidas instâncias do UMCourt no domínio do direito laboral, direito comercial e da família. Neste artigo vamos focar-nos no trabalho desenvolvido no domínio do direito da família e das sucessões, mais especificamente na partilha de bens.

A partilha dos bens comuns consiste no acto através do qual um património deixa de ser indivisível [6]. A partilha conjugal consiste na atribuição definitiva aos cônjuges de sua meação dos bens comuns, enquanto a partilha hereditária consiste na atribuição definitiva dos bens do finado aos seus herdeiros por lei (sucessão legitima) ou por acto de última vontade (sucessão testamentária). A partilha de bens é fonte de conflito quando existe desacordo entre as partes envolvidas sobre a metade que a cada um cabe. O objectivo desta instância é o de dar suporte à definição das partilhas fora dos tribunais, recorrendo à ODR, baseando-se em técnicas de AI e teoria de jogos para obter uma partilha justa, equitativa e satisfatória. Para tal foi explorado o algoritmo Adjusted Winner (AW) desenvolvido por Steven J. Brams e Alan D. Taylor [9] que consiste na divisão de bens entre duas partes da maneira mais justa possível.

3.1 Arquitectura do UMCourt

A estrutura do UMCourt Partilhas está organizada com três componentes principais (fig. 1). O componente AWV (Adjusted Winner by Value) contém os mecanismos e o algoritmo para o processamento da proposta para a partilha de bens. Este componente é responsável pela determinação da BATNA e WATNA de cada parte, neste contexto específico. Os restantes dois componentes, CBR (Case Based Reasoning) e ARG (Argumentation) assentam na arquitectura de agentes previamente definida do UMCourt (figura 2).

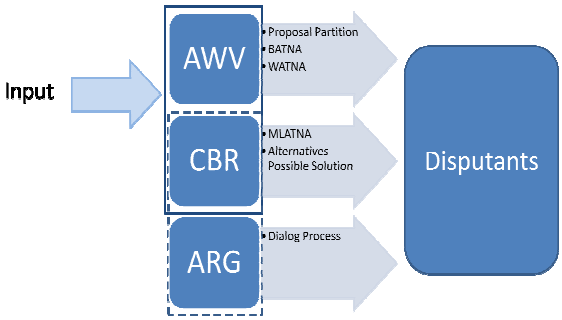


Fig. 1. Estrutura do UMCourt Partilhas

Esta arquitectura é composta por uma série de agentes que disponibilizam serviços específicos que podem ser usados independentemente ou em sequências específicas bem definidas para implementar comportamentos complexos. No âmbito deste artigo vamos apenas fazer uma breve descrição desta arquitectura uma vez que esta se

encontra definida em [13]. Os agentes da arquitectura encontram-se organizados em dois grupos principais: os agentes primários e os agentes secundários. Os agentes primários são caracterizados por ter uma maior autonomia e maiores capacidades de comunicação. Os agentes secundários são apenas responsáveis por suportar a execução dos agentes primários através da prestação de serviços básicos.

Neste sentido, os agentes secundários implementam serviços como a ligação à base de casos, a leitura de casos a partir de ficheiros, a selecção de casos similares, regras que regem o comportamento dos agentes, interligação com agentes externos, entre outros. Estes serviços permitem que os agentes primários implementem comportamentos complexos, nomeadamente o processo de CBR. São também estes serviços genéricos desenvolvidos no âmbito do UMCourt que suportam o funcionamento do UMCourt partilhas, aumentando a reutilização de funcionalidades e simplificando o desenvolvimento.

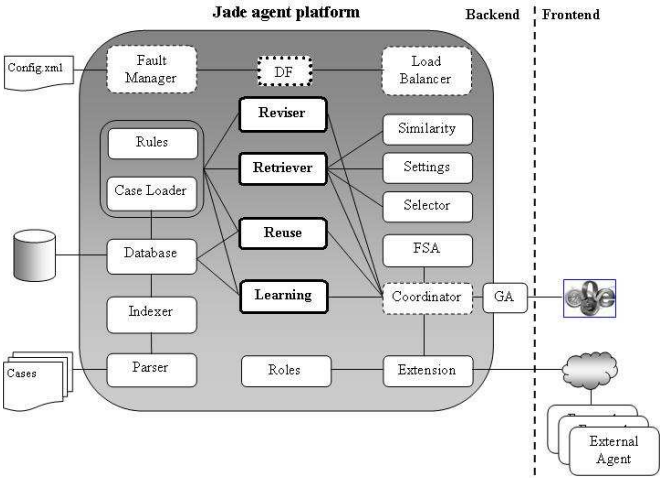


Fig. 2. A arquitectura de base do UMCourt. Os agentes centrais constituem os agentes primários enquanto os restantes constituem agentes secundários.

3.2 Adjusted Winner

O algoritmo Adjusted Winner permite a divisão de um número de itens entre duas partes em conflito. AW utiliza técnicas de teorias de jogos e chega a ilustrar o equilíbrio de Nash. Este algoritmo utiliza a atribuição "secreta" de pontos pelos itens a dividir pelas partes para a divisão dos mesmos. Cada parte deverá alocar um total de 100 pontos pelos itens em causa, o que definirá o seu nível de preferência entre os itens. Os pontos das preferências são de seguida submetidos a uma manipulação matemática que determina a distribuição dos itens pelas partes. Este processo pode ser considerado livre de inveja porque cada parte recebe a meação dos itens de acordo com as preferências atribuídas, i.e. eles recebem os itens ou a metade mais "valiosa" – de acordo com a sua própria avaliação – o que os deixa satisfeitos com a sua metade e não provoca a cobiça para a metade do outro. A divisão é também equitativa porque

cada parte recebe pelo menos 50% dos itens desejados que em alguns casos conseguem mais [7] e elas acreditam que a sua metade vale o mesmo que a da outra parte (tendo em conta os pontos atribuídos).

Considerando o contexto do nosso sistema, vamos exemplificar a partilha de bem para um divórcio utilizando o AW a fim explicar a sua execução. Supondo que Jo e Berta se estão a separar e precisam definir a partilha de bens. O primeiro passo é a definição dos itens ou bens em causa. A seguir cada parte irá distribuir os 100 pontos de preferência entre os bens de acordo com a importância ou valor que cada um atribui aos bens. Suponhamos que a atribuição dos pontos foi feita como apresentado na tabela 1.

Tabela 1. Exemplo da distribuição de pontos

Itens	Jo	Bertha
Vivenda	45	30
Apartamento	20	35
Carro	15	20
Títulos bancários	20	15
Total	100	100

A execução do AW é dividida em duas fases: a fase do vencedor e a fase do ajustamento. A fase do vencedor consiste na alocação do item à parte que lhe atribuiu maior pontuação. Para o nosso caso, nesta fase, a partilha será definida pelo seguinte: Jo ficará com a vivenda e os títulos bancários e Bertha ficará com o apartamento e o carro, totalizando 65 e 55 pontos respectivamente. Desta forma Jo tem mais pontos que Bertha. Quando os pontos adquiridos não são equitativos, existe a necessidade de se transferirem os pontos excedentes da parte que os tem para a parte que deficitária em pontos a fim de equiparar a divisão em pontos. A transferência é feita item a item, o quanto for necessário, até se obter uma divisão equitativa. Esta é a chamada fase do ajustamento. Considera-se o quociente vencedor-perdedor (nº de ponto atribuídos para o item pelo vencedor/ nº de ponto atribuídos para o item pelo por vencedor) para definir a ordem dos itens pela qual os pontos serão transferidos. A ordem é definida pelos quocientes em ordem crescente, indicando os menores valores os itens mais valorizados ou desejado sendo por estes que a transferência deve começar. Assim o quociente da casa é de 45/30 = 1.5 e o dos títulos é de 20/15 = 1.33, e a transferência começa então pelos títulos. A definição da percentagem dos títulos transferidos é feita pelo seguinte:

$$45 + 20p = 35 + 20 + 15(1- p)$$
$$45 + 20p = 70 – 15p$$
$$p = 25/35 \approx 0.714$$

Assim, Jo ficará com a casa e 71.4% dos títulos bancários o que faz um total de 59.285 pontos (45 + [20 * 0.714]), e Bertha ficará com o apartamento, o carro e receberá 28.5% dos títulos o que faz um total de 59.285 pontos (35 + 20 + [15 * 0.285]). Podemos assim ver com este exemplo que as partes vão recebendo os itens de acordo com suas preferências até que totalizam o mesmo número de pontos. Segundo [7], AW é eficiente por não haver uma melhor divisão para as partes, i.e. cada um recebe uma metade acima do esperado (59.285 pontos > 50 pontos) e é equitativa porque cada uma delas recebe exactamente o mesmo numero de pontos.

3.3 Adjusted Winner by Value – Algoritmo para UMCourt Partilhas

UMCourt Partilhas é o protótipo de um sistema de negociação assistida que está a ser desenvolvido para suportar a partilha de bens em casos de divórcios e heranças. O processo utilizado para definir a partilha baseia-se essencialmente no AW e o objectivo é proporcionar uma divisão ainda mais justa. AW é facilmente enquadrado em situações de divórcio por ser uma partilha entre duas entidades. No entanto seu uso para casos de herança já não foi tão linear, visto poder se tratar de uma partilha com duas ou mais partes e nem todas com quotas iguais.

Considerando o resultado acima mencionado, a partilha parece justa por ter sido medida pelos pontos. No entanto, nada garante que as partes são totalmente honestas no acto da alocação de seus pontos. Por exemplo, se uma das partes fizer a alocação considerando o valor monetário dos itens, i.e. atribuindo mais pontos aos itens mais caros e a outra parte não ter noção dos preços dos mesmos ou simplesmente optar pelo critério preferencial (desconhecendo a má vontade da outra parte), em termos monetários esta parte sai a perder. Analisando o exemplo anterior, a divisão aparenta ser de facto justa, porém se for chamado um avaliador para definir o valor monetário dos itens e a divisão ser analisada numa perspectiva monetária, a nossa conclusão pode ser bem diferente. Assumindo que o valor definido de cada item foi: vivenda – 100000, apartamento – 500000, Carro – 30000 e títulos bancários – 70000. De acordo com os itens e respectivas porções recebidas anteriormente, fez-se uma analogia do valor monetário que cada parte irá receber. Assim, Jo que ficou com a totalidade da casa e 71.4% dos títulos bancários, tinha bens num valor aproximado a 149980 e que ficou com a totalidade do apartamento e do carro e 28.5% dos títulos bancários, tinha bens num valor aproximado a 549950. Se pelos pontos a divisão parecia equitativa, considerando o valor monetário dos bens, esta não parece mais.

Para resolver este problema de divisão justa considerando a vertente monetária fizemos algumas alterações ao AW adicionando a componente do valor monetário para cada item, fazendo as manipulações matemáticas nesta vertente. Segue-se uma descrição formal do algoritmo *Adjusted Winner by Value*.

3.3.1 Adjusted Winner by Value para divórcios

Definição do problema. Seja I = $\{i_1, i_2, \ldots, 1_n\}$ o conjunto de n itens com os respectivos valores V=$\{v_1, v_2, \ldots, v_n\}$ que se pretende dividir pelos cônjuges H(usband) e W(ife). H e W fazem a distribuição dos 100 pontos de preferência por cada item i. Assim teremos:

$$HP = \sum Hpi = 100 \text{ e } WP = \sum Wpi = 100 \text{ onde i } \in \{1, 2, \ldots, n\} \quad (1)$$

Onde WP e HP representam os pontos atribuídos por W e H respectivamente.

Fase do vencedor. Nesta fase o procedimento é semelhante ao AW. A atribuição de cada item é feita à parte que maior pontuação tiver sobre o item.

$$\text{Se } Wpi \geq Hpi \text{ então } Wvi = Vi \text{ senão } Hvi = Vi \quad (2)$$

Onde $\sum_{i=1}^{n} Wvi$ e $\sum_{i=1}^{n} Hvi$ representam o valor monetário alocado pelas partes W e H respectivamente, de acordo com os itens recebidos. Para todo item i não recebido, Wvi e Hvi recebem o valor 0.

Fase do ajustamento. Ao contrário de como se procede no AW, a equidade da partilha não é determinada pelos pontos, mas pelos valores monetários dos itens. Assim, depois da alocação dos itens pelas partes, comparam-se os valores monetários que cada um recebeu pela partilha mediante os pontos. Caso o total monetário de ambas partes for igual, consideramos divisão equitativa, caso contrário deve-se proceder a transferência dos valores excedentes da parte avantajada para a parte deficitária. É importante referir que esta transferência é feita mediante as preferências de ambas partes, i.e. é definido o conjunto Q dos quocientes vencedor-perdedor através dos quais será determinada a ordem dos itens transferidos. Assim teremos:

$$\text{Se } \sum Wvi = \sum Hvi \rightarrow \text{Partilha equitativa}$$

$$\text{Senão se } Wv \geq Hv \text{ então } qi = Wpi/Hpi \text{ senão } qi = Hpi/Wpi \quad (3)$$

Ordena-se o conjunto Q por ordem crescente e colocam-se os respectivos itens i no conjunto O para todo $qi \geq 1$. Assim os valores dos itens de O vão sendo transferidos para a parte até que se igualem os valores de ambas partes.

Utilizando o *AW by value* no exemplo acima mencionado, a partilha ficará como se segue: Jo ficará com a totalidade da vivenda, do carro e dos títulos e deverá receber 30% do apartamento. Bertha receberá 70% do apartamento. Cada parte fica assim com bens avaliados em 350000, para uma divisão 50-50. De acordo com os pontos de preferência e o valor monetário esta partilha pode ser considerada justa.

3.3.2 Adjusted Winner by Value para cenários de heranças

Para garantir uma partilha justa e equitativa com o *AW by value* em casos de herança o procedimento é o mesmo tendo a necessidade de adaptar a partilha pelo número de pessoas e ter em conta as quotas da herança que cabe a cada herdeiro.

Assim na fase do vencedor, o item é atribuído ao herdeiro que maior preferência (pontos) exprimir pelo item. No entanto, por ser uma partilha com duas ou mais partes, algumas regras (apoiada na lei) devem ser inseridas para evitar um resultado que provoque inveja. Assim, antes de se começar a distribuição inicial dos bens, devem ser definidos os herdeiros com atribuição de preferência sobre os itens (art. 2103º-A,B e C) e com bens doados por colação (art. 2104º e 2115º). A estes, é-lhes atribuído um grau de primazia sobre o item em causa. No acto da atribuição dos itens às partes, se houver empate nos pontos, o item vai para o herdeiro com maior primazia. Se nenhum dos herdeiros empatados tiver primazia sobre o item, os herdeiros envolvidos no empate perdem o direito sobre ele que passa para o herdeiro com a maior pontuação logo a seguir. O critério da primazia é definido pelos herdeiros, i.e. herdeiros com primazia legal podem solicitar que a atribuição inicial dos itens seja determinada preferencialmente pelos pontos ou directamente pela primazia.

Na fase do ajustamento, verifica-se se cada herdeiro tem o valor monetário de itens equivalente a sua quota. A transferência dos excessos começará pelo herdeiro com maior excedente em relação à sua quota, para o herdeiro com o menor quociente

vencedor-perdedor do item em causa em relação aos outros herdeiros, com défice na sua quota.

4 Demonstração do Caso de Estudo para a Divisão de Bens em Caso de Divórcio

Para o desenvolvimento deste protótipo, ainda não foi preocupação criar interfaces que garantam a privacidade de cada parte por formas a facilitas os seus testes de funcionamento.

Numa primeira etapa as partes são identificadas e é definido o número de itens que pretendem dividir. De seguida, são especificados quais os itens em causa, os seus valores e as preferências de cada parte. A segunda etapa consiste na geração da proposta de divisão dos bens. De acordo com a informação fornecida, são apresentadas na fase do vencedor as respectivas alocações em valores monetários mediante os pontos de preferência. Se o valor inicial não for equitativo passa-se para a fase do ajustamento.

Fazendo alusão ao nosso exemplo, na fase do vencedor, a alocação inicial dos itens pelos valores monetários apresenta uma disparidade onde Jo fica com a vivenda e os títulos bancários num valor total de 170.000 e Bertha fica com o apartamento e o carro num valor total de 530.000. Não sendo equitativo o resultado, na fase do ajustamento, alguns bens de Bertha têm de ser transferidos para Jo por ela possuir bens com maior valor monetário. Assim, suportados pelas preferências inicialmente indicadas, Jo recebe a totalidade do carro e 30% do apartamento equiparando os valores de ambas partes em 350.000. Jo ficará então com a totalidade da vivenda, do carro e dos títulos bancários e 30% do apartamento. Bertha ficará com 70% do apartamento com o valor equivalente aos bens de Jo.

Para este resultado ser considerado livre de inveja é necessário que ambas as partes conheçam os valores dos itens a dividir, fazendo assim uma atribuição de pontos conscientes do trade-off que implica com o valor monetário. Assim, consciente de que a divisão é feita tendo em conta as preferências e o valor monetário dos itens, a atribuição das preferências de Jo e Bertha será provavelmente diferente.

Consideremos um segundo exemplo com a atribuição dos pontos feita mediante ponderação dos valores monetários como segue:

Tabela 2. Nova distribuição de pontos em relação ao valor

Itens	Jo	Bertha	Valores
Vivenda	20	30	100.000
Apartamento	45	40	500.000
Carro	15	20	30.000
Título bancários	20	10	70.000
Total	100	100	700.000

Depois do processamento com os dados acima apresentados, a proposta de divisão é formada pela totalidade da vivenda e do carro e 44% do valor do apartamento para Bertha, e para Jo, a totalidade dos títulos bancários e 56% do valor do apartamento.

Ambos ficam com os bens de acordo com suas preferências no valor de 350.000. Esta partilha é considerada equitativa, por ambas as partes receberem 50% dos bens; livre de inveja, por cada parte ir recebendo os bens mediante suas próprias preferências; e eficiente, pelo facto de não haver outra partilha equitativa e livre de inveja considerando os mesmos parâmetros.

5 Trabalho Futuro

Embora a divisão apresentada pelo *AW by value* seja considerada equitativa e justa, monetária e preferencialmente, as partes podem não aceitá-la tal como é proposta e para resolver a questão pode ser necessário criar-se mecanismos de negociação e apresentar outras alternativas. A constituição de outras alternativas conta com a exploração do raciocínio baseado em casos (CBR[7]). O CBR pode ser descrito como uma metodologia para resolução de problemas que se baseia em experiencias e conhecimentos passados para a tomada de decisões presentes [11]. O uso do CBR é muito usual no comportamento humano, consciente ou inconscientemente. Na área da lei também é comummente identificada esta metodologia, onde para estimar ou definir uma sentença, os advogados ou juízes recorrem a casos anteriores fazendo analogias ao caso presente. O CBR é também conhecido como uma das técnicas da AI capaz de utilizar conhecimento previamente adquirido para resolução de questões e problemas concretos (casos), assim novos casos são resolvidos pelo conhecimento e reutilização de casos similares anteriores [12]. Explorando a combinação do CBR na área da AI e da lei, o UMCourt Partilhas pretende auxiliar e suportar a negociação entre as partes.

A fim de melhor o fazer considerou-se essencial fornecer-se as respectivas BATNA e WATNA a cada parte. Sendo definidos estes extremos para cada parte e considerando que frequentemente a BATNA de uma parte é a WATNA da outra, é delimitada de forma menos custosa a ZOPA – zona de possível acordo (Zone of Possible Agreement) [13] (Fig. 3). O componente CBR que contém a base de casos e os mecanismos que permitem a recuperação, reutilização, revisão ou correcção e a actualização da informação contida na base, ou seja a manipulação dos casos. Considera-se ainda a determinação da MLATNA - mais provável alternativa para um acordo negociado (Most Likely Alternative to a Negotiated Agreement). A MLATNA é calculada utilizando a probabilidade de ocorrência dos casos similares da base.

O suporte para a argumentação está a ser implementado no componente ARG que contém os mecanismos que possibilitam a troca de argumentos entre as partes no acto da negociação propriamente dita, a fim de permitir as partes de argumentarem e convencer a outra a mudar de posição. Através do ARG serão disponibilizadas ferramentas capazes de suportar todo processo de negociação organizado de acordo com o contexto de contestação e o tipo de diálogo.

O processo de suporte à negociação, que se encontra neste momento em desenvolvimento, decorre em duas etapas: (1) Alimentação e propostas – que engloba os componentes AWV e CBR e (2) Diálogo e negociação. Na primeira etapa o sistema apresenta as propostas de soluções mediante as entradas apresentadas por

[7] Do Inglês – Case Base Reasoning

cada parte, e os prováveis limites de acordos. Na segunda etapa, o sistema disponibiliza mecanismos para o diálogo entre as partes e sempre que necessário volta a calcular possíveis soluções, pelos desacordos ocorridos. Este trabalho que está agora a ser desenvolvido visa usar o CBR para aumentar a eficiência dos processos de negociação e consequentemente do processo de ODR.

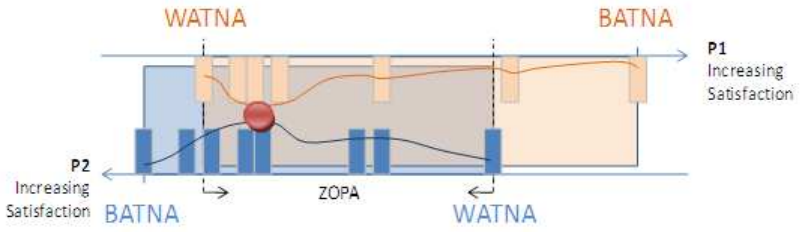


Fig. 3. Representação dos possíveis resultados para cada parte. (Fonte: [13])

6 Conclusões

Os sistemas ODR têm se revelado muito úteis para a resolução de conflitos. Cada sistema apoia áreas específicas e a tendência é o uso da inteligência artificial para torna-los cada vez mais autónomos, com capacidade de apresentar estratégias próprias. As pesquisas na área da inteligência artificial para ODR estão longe de estar esgotadas, mas cremos que os primeiros passos já foram dados. A inteligência artificial possui diversas técnicas e acreditamos que algumas delas precisam de ser combinadas para a obtenção de resultados eficientes.

A justiça da partilha de bens feita com o *AW by value* assenta na importância que cada parte atribui aos itens e nos seus respectivos valores monetários. Embora o sistema apresenta uma proposta de divisão, a decisão final cabe às partes. No exemplo mencionado no texto, Bertha pode querer ficar com o carro apesar de monetariamente ficar com um valor superior à de sua metade, para isso seria necessário compensar Jo de outra forma. Daí a necessidade de implementar um mecanismo que permita às partes de negociar e argumentar possíveis ajustes. A proposta de partilha apresentada pelo sistema serviria de ponto de partida para a negociação em torno dos pontos de discórdia.

Como foi acima referido, para melhor apoiar as partes, um sistema de suporte a negociação precisa fornecer-lhes informações sobre sua posição e possíveis resultados no conflito em causa, e parece-nos relevante a definição e apresentação do BATNA pelo sistema para cada parte. Acreditamos que o BATNA ajudará as partes a reconhecer uma partilha vantajosa e a se concentrarem nas suas melhores alternativas para chegarem ao acordo. O CBR é a técnica da AI que está a ser explorada para enriquecer o processo de negociação utilizando conhecimentos anteriores para fornecer informações mais precisas.

Agradecimentos

O trabalho descrito neste artigo foi suportado pelo projecto TIARAC - *Telematics and Artificial Intelligence in Alternative Conflict Resolution* Project (PTDC/JUR/71354/2006).

Referências

1. Fiadjoe, Albert. Alternative Dispute Resolution: A Developing World Perspective, Routledge-Cavendish (2004)
2. POBLET, M. 2008. Introduction: Bringing a new vision to online dispute resolution. In Expanding the Horizons of ODR, Proceedings of the 5th International Workshop on Online Dispute Resolution (Workshop 08), M. Poblet, Ed. CEUR-Workshop Proceedinkg Series, vol. Volume 430. 1_7.
3. Carneiro D., Novais P., Andrade F. 2009. Artificial Intelligence in Online Dispute Resolution, Relatório Técnico TIARAC Project, Universidade do Minho.
4. Peruginelli, G. 2002. Artificial Intelligence in Alternative Dispute Resolution. In Convegno Lea Workshop 2002. Rapporto tecnico n. 18/2002.
5. Cáceres, E. 2008. EXPERTIUS: A Mexican Judicial Decision-Support System in the Field of Family law. In Francesconi, E. B. E., Sartor, G., & Tiscornia, D. (Eds.), Legal Knowledge and Information Systems (pp. 78-87). IOS Press.
6. Esperança Pereira Mealha, Acordos Conjugais para partilha de bens comuns, 2004, Livraria Almedina
7. Brams, Steven J. (2006). "Fair Division." In Barry R. Weingast and Donald Wittman (eds.), Oxford Handbook of Political Economy. Oxford, UK: Oxford University Press, pp. 425-437.
8. Fisher, R., Patton, B., and Ury,W. 1981. Getting to Yes: Negotiating Agreement Without Giving In. Boston.
9. Brams, S.J. and Taylor, A.D. 1996. Fair Division: From cake cutting to dispute resolution. Cambridge University Press.
10. Zeleznikow, J. and Bellucci, E. 2003. Family Winner: Integrating Game Theory and Heuristics to Provide Negotiation Support. Proceedings of sixteenth International Conference on Legal Knowledge Based Systems, IOS Publications, Amsterdam, Netherlands: 21-30
11. Kolodner, J. L. (1992). An introduction to case-based reasoning. Artificial Intelligence Review, 6(1), 3-34. doi: 10.1007/BF00155578.
12. A. Aamodt, E. Plaza (1994); Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications. IOS Press, Vol. 7: 1, pp. 39-59.
13. Andrade F., Novais P., Carneiro D., Zeleznikow J., Neves J., Using BATNAs and WATNAs in Online Dispute Resolution, in JURISIN 2009 - Third International Workshop on Juris-informatics, Tokyo, Japan, ISBN 4-915905-38-1, pp 15-26, 2009.
14. De Vries BR., Leenes, R., Zeleznikow, J., Fundamentals of providing negotiation support online: the need for developping BATNAs. Proceedings of the Second International ODR Workshop, Tilburg, Wolf Legal Publishers (2005) 59-67.

Sistema Inteligente de Pesquisa de Eventos em Enfermagem

António Morais, José Machado, António Abelha, and José Neves

Departamento de Informática, Universidade do Minho,
Braga, Portugal
A44636@alunos.uminho.pt,
{jmac,abelha,jneves}@di.uminho.pt
http://www.di.uminho.pt/

Resumo Actualmente, a qualidade dos cuidados de saúde é uma prioridade. Para tal, as Instituições de Saúde têm de adquirir práticas e sistemas de controlo capazes de aumentarem a qualidade dos seus serviços. As Instituições têm ao seu dispor um conjunto de indicadores. Estes indicadores dividem-se segundo os serviços e eventos que se pretendem avaliar. Na área da enfermagem foi dada especial atenção aos indicadores de queda e úlcera de pressão. Para avaliar estes indicadores foi criado o Sistema de Pesquisa de Eventos em Enfermagem. Este sistema é composto por um conjunto de procedimentos PL/SQL e por uma interface *Web*. Os resultados obtidos pelo sistema foram os esperados, sendo iguais aos provenientes das consultas SQL previamente utilizadas pelos profissionais de saúde. A obtenção destes indicadores torna-se, assim, mais rápida e liberta recursos quando comparada com a anterior. O sistema desenvolvido é facilmente expansível para outros indicadores e parâmetros de pesquisa.

Keywords: Enfermagem, Indicadores, Queda, Úlcera de Pressão, Sistema de Pesquisa

Abstract. Currently, the healthcare quality is a priority. To this end, the Healthcare Facilities must acquire practices and control systems capable of increasing the quality of their services. The institutions have at their disposal a set of indicators. These indicators are divided according to the services and events which are to be assessed. In the nursing field was given special attention to falls and pressure ulcers indicators. To assess these indicators an Event Search System in Nursing was created. This system consists on a set of PL/SQL procedures and a Web interface. The system results were the expected, being equal to those previously obtained from the SQL queries used by healthcare professionals. Thus, achieving these indicators becomes more quickly and frees resources when compared with the previous method. The developed system is easily expandable to other indicators and search parameters.

Keywords: Nursing, Indicators, Fall, Pressure Ulcers, Search System

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 791–802

1 Introdução

A qualidade dos cuidados de saúde tornou-se um assunto de grande debate dentro e fora das Instituições de Saúde (IS). Muito deste interesse na qualidade dos cuidados de saúde cresceu devido às recentes transformações dos sistemas de saúde, acompanhadas de novas estruturas e estratégias organizacionais que afectam a qualidade do atendimento. Todavia, ainda existem falhas no que respeita à recolha sistemática de informação nos sistemas de saúde capaz de produzir conhecimento relativo à qualidade dos cuidados prestados. Para a colmatação dessas falhas torna-se necessário analisar a seguinte questão: o que se sabe sobre a qualidade dos cuidados de saúde? Através de uma análise da literatura [1,2,3,4] constata-se que existe uma grande falta de documentação sobre a forma como são tratadas as principais doenças e episódios na maioria dos sistemas de cuidados de saúde; uma falta de avaliação de resultados sistemáticos; uma falta de avaliação dos recursos relacionados à qualidade de cuidados e situações específicas que ocorrem nos sistemas de saúde, persistindo variações entre prestadores de cuidados a pacientes similares; e, por fim, a não existência de sistemas de controlo em vigor nas instituições prestadoras de cuidados de saúde ou reguladoras. Existem ainda dificuldades adicionais, como a falta de conhecimento e interesse em muitos países sobre os problemas relacionados com a qualidade dos seus serviços e a sua potencial interferência na melhoria e credibilidade da vida das IS. Casos como estes vêm impor entraves à implementação de sistemas capazes de produzir informação acerca da qualidade das IS e dos seus serviços. O acesso a padrões de qualidade tornou-se muito importante para as IS, organizações reguladoras e para os próprios utentes. Cada vez mais, os utentes começam a exigir às IS indicadores de qualidade e diferenciação relativamente aos serviços por elas fornecidos, dando bastante ênfase à relação custo-eficiência dos cuidados prestados. Os indicadores de performance permitem fazer a avaliação da qualidade dos cuidados e serviços de saúde. Esta avaliação pode ser feita através da criação de indicadores de qualidade que descrevem o desempenho para um determinado tipo de cuidado de saúde e permitem avaliar se está de acordo com os indicadores standards dos cuidados de saúde [5].
A qualidade dos cuidados de saúde pode ser definida como "o grau em que os serviços de saúde de indivíduos e populações aumentam a probabilidade de resultados de saúde desejados e são consistentes com o conhecimento profissional actual" [6] e pode ser dividida em diferentes dimensões de acordo com os cuidados a ser avaliados [7]. Os indicadores podem ser definidos de diversas formas: como medidas que avaliam um processo especial dos cuidados de saúde ou o seu resultado [8]; como medidas quantitativas que podem ser usadas para monitorizar e avaliar a qualidade da administração e gestão, prestação de cuidados e funções de suporte que afectam os pacientes [9]; e, como instrumentos de medição, monitorização, ou alerta que são utilizados como guias para monitorizar, avaliar e melhorar a qualidade dos cuidados dos pacientes, serviços de apoio clínico e função organizacional que afectam os pacientes [10].

Os indicadores fornecem uma base quantitativa para os profissionais de saúde e organizações atingirem os objectivos de melhorarem os cuidados de saúde e

os processos pelos quais estes são fornecidos. A monitorização e medição dos indicadores permite atingir muitos propósitos. Os indicadores permitem: a documentação da qualidade dos cuidados de saúde; a possibilidade de comparação de resultados ao longo do tempo e entre instituições; definir prioridades e tomar decisões (e.g. a escolha de uma IS ou mesmo de um profissional de saúde); a responsabilização, regulamentação e acreditação; a possibilidade de melhoria da qualidade; e, o suporte para a escolha dos prestadores de saúde por parte do paciente. O uso de indicadores permite aos profissionais de saúde e às organizações monitorizar e avaliar o que acontece aos seus pacientes em função da forma como são prestados os seus serviços. Porém, ao contrário do que pode acontecer é errado pensar que os indicadores são uma avaliação directa da qualidade dos serviços e da própria IS. O conceito de qualidade é algo multidimensional, pelo que compreender e avaliar este conceito requer várias análises distintas.
Os indicadores são baseados em standards da área da saúde. Estes podem ser baseados em provas e derivar da literatura ou, então, quando é verificada a necessidade da existência ou criação de um indicador, este pode ser determinado através de um grupo de profissionais de saúde de acordo com a sua experiência. Assim sendo, os indicadores e standards podem ser descritos de acordo com a sua importância para obter e prever resultados relevantes [11]. Os indicadores encontram-se divididos em vários grupos abrangendo um elevado número de funcionalidades das IS. No entanto, no presente estudo apenas interessam os indicadores referentes às taxas de risco de acontecimento de um fenómeno, como quedas e úlceras de pressão. Após verificar-se o correcto tratamento destes fenómenos por parte do sistema criado, será extremamente fácil e rápido o seu alargamento aos restantes fenómenos. A escolha destes fenómenos prende-se com a urgência que existe em travar estes acontecimentos dentro das IS, uma vez que têm uma elevada taxa de ocorrência.

2 Risco de Queda

A existência de quedas nas IS é considerado um sério problema de saúde. As quedas de pacientes não só levam ao aumento dos custos de saúde como também interferem gravemente na qualidade de vida dos pacientes e impedem a sua independência ambulatória [13]. As consequências advindas das quedas incluem lesões físicas e traumas emocionais, podendo atingir valores mais graves no caso de pacientes com idades elevadas, como a própria morte. Torna-se assim vantajoso o investimento de conhecimento e recursos para identificar os pacientes com risco de sofrerem quedas e implementar um sistema compreensivo de educação e prevenção desse acontecimento. As quedas de pacientes são um dos eventos adversos registados mais comuns nas IS [14,15]. Estudos previamente realizados referem que estes eventos representam cerca de 40% dos incidentes relatados com pacientes internados e ocorrem em mais de 7% das admissões hospitalares [16,17,18,19]. Mais de um terço das quedas de pacientes internados resulta em uma ou mais lesões [20,16,21,17,19]. Enquanto que a maioria das lesões são insignificantes (por exemplo, abrasões, lacerações, hematomas, e con-

tusões), aproximadamente 3% das quedas resultam em fracturas [16,21,17,19]. Devido à frequência de quedas e à associação de morbidez, as IS começaram a incentivar o desenvolvimento de sistemas e politicas de prevenção de quedas [14]. Desde 2005, a *Joint Commission for Accreditation of Healthcare Organizations* (JCAHO) informou que a prevenção de quedas é um dos objectivos da *National Patient Safety*. De acordo com estes valores é de fulcral importância que as IS comecem a adquirir estratégias e sistemas capazes de travar estes resultados. Porém, mais importante que a sua existência é a sua adequação à complexidade das base de dados das IS onde a informação se encontra armazenada.
Associado ao evento queda existem várias formas de cálculo de indicadores, sendo que algumas referem-se à razão da ocorrência do evento em relação à ocorrência dos restantes eventos na IS, e outras à capacidade de prevenção da ocorrência desse evento, denominado de taxa de eficácia na prevenção de quedas. O cálculo da taxa de eficácia na prevenção de quedas é determinado através da seguinte fórmula:

$$\textbf{TEPQ}^{1}=\frac{\text{nº episódios de queda, com risco prévio}}{\text{nº total de episódios de risco de queda}}[22], \tag{1}$$

Através desta fórmula pretende-se averiguar a taxa de eficácia na prevenção de quedas, através da razão entre os registos com risco prévio da ocorrência de queda e o número total de registos de risco de quedas registadas no mesmo período. É muito importante ter em conta o aspecto anteriormente referido, pois para o numerador apenas se pode ter em conta os casos de ocorrência de queda que tiveram associado previamente um registo de risco da sua ocorrência. Esses são os casos que importam para poder medir com eficácia a capacidade da IS em dar resposta às necessidades dos seus pacientes. Neste caso, a necessidade refere-se à capacidade de não ocorrerem episódios de queda sabendo que o paciente tem uma possibilidade elevada de que isso aconteça.

3 Risco de desenvolvimento de Úlceras de Pressão

Tal como os episódios de queda o desenvolvimento de úlceras de pressão é bastante comum nas IS [23]. A importância do seu tratamento e da melhoria da qualidade dos cuidados de saúde onde este fenómeno ocorre levou a que associações como a *Hospital Quality Alliance* e *England's National Health Service* (NHS) propusessem como um indicador chave da prestação dos cuidados de saúde [24].
No Reino Unido estudos concluíram que a prevalência de úlceras de pressão em pacientes hospitalizados situava-se entre os 9.6% e os 11.90% em 2007, sendo que em pacientes acamados o seu valor subia para 12% e em paciente idosos o seu valor atingia o máximo de 22.07% [25]. O tratamento destes episódios requer o

[1] Taxa de Eficácia na Prevenção de Quedas.

uso de recursos (equipamento e profissionais de saúde) da IS, para além de ser dispendioso. No Reino Unido a *National Health Service* (NHS) estima que anualmente sejam gastos nestes cuidados cerca de 1.7 a 2.1 biliões de euros. Tendo um custo de tratamento por paciente de 1 273 euros em casos de úlceras de grau 1 e 9 275 euros no caso de úlceras de grau 4 [26]. Muitos destes casos podem ser evitados, uma vez, que ocorrem como resultado de negligências por parte das IS. Devido a situações como estas as associações reguladoras da saúde começaram a pressionar as IS para adquirirem sistemas capazes de ajudar na prevenção de fenómenos como as úlceras de pressão.
No sistema desenvolvido foi dada especial atenção a este indicador, sendo um dos primeiros a ser incluído no sistema. A fórmula de cálculo utilizada para o cálculo do risco de desenvolvimento de úlcera de pressão é similar à utilizada para o cálculo de risco de queda.

$$\textbf{TEPUP}^{2}=\frac{\text{nº episódios de úlceras de pressão, com risco prévio}}{\text{nº total de episódios de risco de úlcera de pressão}}\text{[22].} \qquad (2)$$

4 Desenvolvimento do Sistema Inteligente de Pesquisa de Eventos em Enfermagem - SIPEE

O SIPEE foi pensado e desenvolvido de forma a que possa corrigir erros ou incoerências de registos. A inserção da informação nas BD hospitalares está muitas vezes sujeitas a situações de incoerência. A incoerência de registos pode ocorrer no registo da especificação do acontecimento (queda, risco de queda, úlcera de pressão, risco de úlcera de pressão), uma vez que este campo é de texto livre. Para resolver estes problemas no sistema, introduziram-se regras de validação que fazem a associação dos registos efectuados pelos utilizadores com os indicadores de qualidade. Desta forma, quando um utilizador acrescenta um novo episódio de internamento com uma especificação referente a uma queda ou úlcera de pressão o sistema faz a sua validação e associa esse episódio ao respectivo indicador, tendo em conta se ele corresponde a um caso de risco de acontecimento ou apenas acontecimento. Através deste tipo de validação salvaguardam-se incoerências de registo de dados e de deturpação de resultados finais dos indicadores, conferindo ao sistema um carácter inteligente.
O sistema desenvolvido consiste num motor de pesquisa e numa interface gráfica de apresentação e interacção com o sistema. Para o desenvolvimento do motor de pesquisa do SIPEE recorreu-se ao uso da linguagem PL/SQL.

4.1 PL/SQL

A informação necessária para o tratamento e obtenção dos valores de medição associada aos indicadores de qualidade necessita de ser tratada e posteriormente

[2] Taxa de Eficácia na Prevenção de Úlceras de Pressão.

submetida a um sistema capaz de obter esses valores. Desta forma, torna-se imprescindível recorrer ao uso de uma linguagem com capacidade para a execução de instruções directas sobre os dados. Para atingir tal objectivo recorreu-se ao uso da linguagem PL/SQL (Procedural Language/Structured Query Language). Esta linguagem traz enormes vantagens e permite a manipulação eficiente dos dados armazenados [12]. A linguagem PL/SQL surge como uma ampliação à linguagem SQL incluindo características das linguagens de programação e mantendo possível a manipulação de dados e instruções de consulta SQL dentro das unidades processuais do código criado [12].

PL/SQL é uma linguagem processual desenvolvida pela Oracle. Esta linguagem veio trazer recursos de engenharia de *software*, tais como o encapsulamento de dados, manipulação de excepções e orientação a objectos. PL/SQL incorpora muitos dos recursos avançados feitos em linguagens de programação concebidas durante os anos 1970 e 1980. Permite a manipulação de dados e a inclusão de instruções de consulta de SQL no bloco de estruturas e unidades processuais do código, tornando a linguagem PL/SQL numa linguagem de processamento de transacções. Com PL/SQL, podem-se usar instruções SQL para limpeza de dados e declarações de controle PL/SQL para processar os dados [12].

4.2 SIPEE

O SIPEE foi desenvolvido com o intuito de ser testado no Centro Hospitalar do Tâmega e Sousa (CHTS), desta forma foi necessário primeiramente estudar quais as tabelas com interesse existentes na BD de registos de enfermagem do CHTS e, mais importante ainda, compreender a forma como estas se relacionam. De entre todas as tabelas presentes na IS apenas foram necessárias as tabelas *status* e *fenomenos*, presentes no *schema enfin* e a tabela *int_transferencias* pertencente ao *schema sgd*. As tabelas anteriores relacionam-se como representado na figura 1.

O motor do sistema tem como função proceder à pesquisa de eventos segundo os parâmetros que lhe são fornecidos através da interface. Para qualquer evento pretendido a forma de pesquisa é sempre igual. Esta consiste em calcular dois valores: o numerador e o denominador. O numerador corresponde ao número de casos ocorridos em que foi declarado no sistema o seu risco de ocorrência. O denominador corresponde ao número total de casos ocorridos no mesmo período em que foi declarado risco de ocorrência.
O motor de pesquisa é composto por um conjunto de quatro procedimentos: dois para o cálculo do numerador e do denominador, *indice_mes* e *indice_ano* (um referente à pesquisa por mês e outro referente à pesquisa por ano, respectivamente), e dois para a selecção dos casos que estão presentes no denominador e não se encontram no numerador, *denominador_mes* e *denominador_ano*, também eles referentes à pesquisa por mês e por ano. Os procedimentos de pesquisa por mês e por ano são em tudo idênticos apenas diferenciando-se na pesquisa do parâmetro data de ocorrência (to_char(data, 'yyyymm') = 'anomes' e to_char(data, 'yyyy') = 'ano', respectivamente). No caso dos procedimentos *indice_mes* e *indice_ano*, primeiramente é feito um *select* desse evento na tabela *status*, tendo em conta

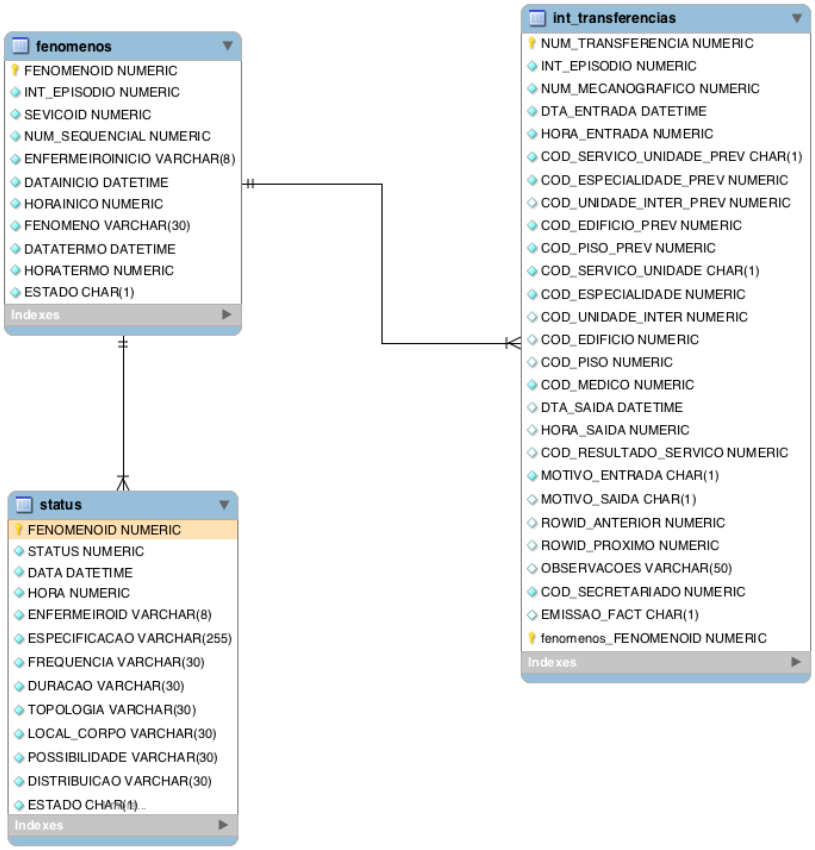


Figura 1. Esquema parcial da base de dados de registos de enfermagem do CHTS.

que a data de ocorrência do evento tem de ser superior à data em que foi declarado o risco da sua existência. No caso de terem ocorrido as duas declarações, risco e evento, na mesma data tem de se efectuar o mesmo raciocínio em relação à hora, ou seja:

Condição 1

$$\text{fenomeno.data} > \text{fenomeno.data\_risco}$$
$$\text{e fenomeno.hora} > \text{fenomeno.hora\_risco}.$$

Nos casos em que a condição anterior se verifica é então usado o parâmetro *fenomenoid* para aceder ao valor correspondente ao episódio de internamento (*int_episodio*) na tabela *fenomenos*. Desta forma, através do episódio de internamento é possível fazer uma pesquisa diferenciada tendo em conta a especialidade e a unidade em que o evento ocorreu. Esta pesquisa é, então, feita na tabela *int_transferencias* e necessita de ter em conta dois aspectos muito importantes.

Tal como na pesquisa inicial, no caso da pesquisa do evento por especialidades e unidades a data do acontecimento tem de ser superior à data de entrada do paciente na especialidade ou unidade e tem, também, de ser inferior a sua data de saída na respectiva especialidade ou unidade. O mesmo acontece para o campo hora. A segunda condição é que data de saída do paciente de uma especialidade ou unidade nunca pode ser nula. Assim sendo, as duas condições que têm de se verificar sempre são:

Condição 2

data_entrada >fenomeno.data >data_saida
e hora_entrada >fenomeno.hora >hora_saida

Condição 3

data_saida is not null.

Desta forma, conseguem-se obter os valores pertencentes ao numerador da equação.
O passo seguinte é o cálculo do denominador. Este é calculado de forma idêntica ao numerador, diferenciando-se apenas no passo inicial que neste caso não é necessário. Neste ponto os episódios de internamento já estão agrupados segundo as respectivas especialidades e unidades onde ocorreram, sendo apenas necessário fazer a sua contagem. Esta contagem é feita quer para os casos presentes no numerador quer para os casos presentes no denominador.
A fase final consiste na apresentação destes valores na interface do sistema.

A informação é apresentada na forma de uma tabela onde pode ser visualizada para cada especialidade e unidade o respectivo total de eventos ocorridos. É, ainda, possível consultar as informações detalhadas dos eventos ocorridos que não foram declarados no numerador.

Para a apresentação destas informações são utilizados os procedimentos *denominador_mes* e *denominador_ano*, dependendo se a pesquisa foi feita por mês ou por ano, respectivamente. Estes procedimentos utilizam a informação gerada pelos procedimentos *indice_mes* e *indice_ano*, respectivamente, que é gravada em duas tabela temporárias. Esta informação está dividida nas tabelas em casos pertencentes ao numerador (tabela temporária numerador) e em casos pertencentes ao denominador (tabela temporária denominador). Os procedimentos anteriores apenas usam essa informação para seleccionar os casos da tabela denominador que não estão presentes na tabela numerador. De referir que como estas tabelas são temporárias sempre que o sistema fecha a ligação à base de dados elas são eliminadas. Isto torna-se bastante vantajoso visto que não fica a ocupar espaço desnecessário em disco.

5 Resultados

O sistema desenvolvido foi submetido a testes com dados de uma IS Portuguesa (CHTS) e o seu resultado foi comparado com as consultas SQL previamente

utilizadas pelos profissionais de saúde para obterem esse mesmos dados.
Para a comparação de resultados foi tido em conta que para a obtenção de cada uma das taxas de eficácia utilizadas pelos profissionais de saúde são necessários 4 passos:

1. Abrir ficheiro onde se encontram as consultas SQL;
2. Copiar as consultas SQL para cálculo do numerador e do denominador;
3. Executar as consultas SQL;
4. Calcular a taxa de eficácia.

Em relação ao SIPEE apenas é necessário inserir qual o fenómeno que se pretende pesquisar e o intervalo de tempo, consistindo assim num conjunto de 2 passos.
Os resultados obtidos, para o numerador e denominador, através dos dois métodos foram iguais comprovando a veracidade do sistema desenvolvido. Porém, com a utilização do sistema a pesquisa de indicadores torna-se muito mais eficaz e intuitiva, poupando tempo e recursos. Os dois métodos foram testados com pesquisas para os indicadores: **risco de queda** e **risco de úlceras de pressão**, obtendo os seguintes resultados.

Indicador **risco de queda**

Para os parâmetros Fenómeno=queda; Ano=2009 e Mês=01, os resultados obtidos foram os apresentados na tabela 1.

Tabela 1. Resultados para o indicador queda.

	Ensaio (tempo(s))					
Método	1	2	3	4	5	**Média(s)**
SIPEE	10.7	10.0	10.0	11.4	09.6	**10.3**
SQL	22.5	20.4	21.6	23.3	22.0	**22.1**

Após a comparação dos valores obtidos pelos dois métodos é possível constatar que o tempo necessário para o cálculo da taxa de eficácia na prevenção do risco demora, segundo o método tradicional, em média mais do dobro do que através do SIPEE.
Utilizando os parâmetros Fenómeno=queda; Ano=2009, ou seja, calculando os tempos dos dois métodos para o ano inteiro, obtém-se os valores apresentados na tabela 2.

Mais uma vez após a comparação dos valores provenientes das consultas SQL com os valores do sistema chegou-se à conclusão que o SIPEE é mais rápido

Tabela 2. Resultados para o indicador queda.

	Ensaio (tempo(s))					
Método	1	2	3	4	5	**Média(s)**
SIPEE	23.6	14.1	13.2	13.0	13.4	**20.2**
SQL	34.0	32.6	31.0	32.8	33.2	**32.8**

para executar o cálculo da taxa de eficácia na prevenção de quedas para o ano inteiro.

Indicador **risco de úlcera de pressão**

Para os parâmetros Fenómeno=úlcera de pressão; Ano=2009 e Mês=01, os resultados obtidos para os dois métodos foram os representados na tabela 3.

Tabela 3. Resultados para o indicador úlcera de pressão.

	Ensaio (tempo(s))					
Método	1	2	3	4	5	**Média(s)**
SIPEE	15.0	14.7	13.0	15.4	14.3	**14.5**
SQL	25.4	27.2	28.7	27.3	26.3	**27.0**

Utilizando a pesquisa do parâmetro úlcera de pressão mas para todo o ano de 2009 os resultados obtidos foram os apresentados na tabela 4.

Tabela 4. Resultados para o indicador úlcera de pressão.

	Ensaio (tempo(s))					
Método	1	2	3	4	5	**Média(s)**
SIPEE	16.9	15.6	14.8	12.7	17.3	**15.5**
SQL	33.1	28.2	31.6	30.3	30.8	**30.8**

Através dos resultados obtidos anteriormente é possível concluir que o SIPEE é mais eficaz no cálculo das taxas de prevenção de eficácia de quedas e úlceras de pressão.

6 Conclusões e Trabalho Futuro

Embora, neste momento o sistema desenvolvido apenas permita calcular os indicadores de risco de queda e risco de úlcera de pressão, uma vez que se encontra

na fase de teste, este corresponde em todo ao que tinha sido inicialmente programado. O sistema é capaz de fazer a pesquisa dos indicadores de risco de queda e risco de úlcera de pressão e devolver os seus resultados de acordo com os parâmetros pretendidos. A utilização deste sistema vem poupar tempo e recursos para a obtenção destes valores. Desta forma, não é necessário estar a correr as consultas SQL directamente cada vez que se pretende obter estes indicadores. Embora as diferenças se situem na ordem de segundos entre os dois métodos de cálculo em termos de recursos o SIPEE traz mais vantagens. Qualquer pessoa com o mínimo conhecimento sobre indicadores de enfermagem e capacidade de manuseamento de um computador é capaz de obter os valores para os indicadores pretendidos, não sendo necessário recorrer ao uso de técnicos especializados para procederem ao cálculo dos indicadores.

A forma como o sistema se encontra estruturado permite que a sua integração com os restantes indicadores de enfermagem seja bastante simples, dado que estes funcionam na mesma base dos desenvolvidos. O sistema está também capacitado para ser constantemente actualizado no que respeita à adição de indicadores, uma vez que apenas é necessário adicionar o procedimento PL/SQL para o cálculo do mesmo.

Contudo, o sistema ainda apresenta algumas limitações, nomeadamente no que se refere ao aspecto de apresentação dos dados. A apresentação das especialidades e unidades na interface ainda é feita através do código interno da IS não havendo a sua conversão para o nome normalmente utilizado para a designar. Isto traz alguns inconvenientes uma vez que torna-se menos intuitivo saber qual a unidade ou especialidade a que o código se refere, levando a que seja necessário estar sempre a consultar uma tabela de conversão. Contudo, esta limitação é fácil de resolver bastando para isso ter acesso à tabela que permite fazer a conversão entre o código e o nome da unidade e especialidade e adicionar nos procedimentos essa conversão.

Referências

1. Schuster M., McGlynn E., Brook R.: How good is the quality of health care in The United States?. Milbank Q. 76, 517–563 (1998)
2. Chassin M., Galvin R.: The urgent need to improve health care quality. Institute of Medicine National Roundtable on Health Care Quality. J Am Med Assoc. 280, 1000—1005 (1998)
3. President's Advisory Commission on Consumer Protection and Quality First. Better Health Care for All Americans. Final Report to the President of the United States. Washington, DC: President's Advisory Commission on Consumer Protection and Quality First (2000)
4. Mainz J., Bartels P., Laustsen S. et al. The National Indicator Project for monitoring and improving medical technical care. Ugeskr Laeger. 163, 6401—6406 (2001)
5. Mainz, J.: Defining and classifying clinical indicators for quality improvement. International Journal for Quality in Health Care. Volume 15. Number 6, 523—530 (2003)
6. Lohr, KN.:Kesselman, C.: Medicare: A Strategy for Quality Assurance.Vols I and II. Morgan Kaufmann. National Academy Press, Washington, DC (1990)

7. Donabedian, A.: The quality of medical care. Science. 200, 856—864 (1987)
8. Worning, A.M., Mainz, J., Klazinga, N., Gotrik, JK., Johansen, K.S.: Policy on quality development for the medical profession. Ugeskr Laeger. 154, 3523—3533 (1992)
9. JCAHO.: Characteristics of clinical indicators. Qual. Rev. Bull. 11, 330–339 (1989)
10. Canadian Council on Health Services Accreditation.:A guide to the development and use of performance indicators. Canadian Council on Health Services Accreditation, Ottawa (1996)
11. Mainz, J.: Developing clinical indicators. Int. J. Qual. Health Care. 15, i5—i11 (2003)
12. Oracle, `http://www.oracle.com/technology/tech/pl_sql/`
13. Morse, J., Morse, R., Tylko, S.: Development of a scale to identify the fall-prone patients. Canadian Journal on Aging. 8 (4), 366—377 (1989)
14. Oliver, D.: Assessing the risk of falls in hospitals: time for a re-think?. Can. J. Nurs. Res. 38, 89–94 (2006)
15. Sutton, J.C., Standen, P.J., Wallacem W.A.: Patient accidents in hospital: incidence, documentation and significance. Br. J. Clin. Pract. 48, 6–63 (1994)
16. Halfon, P., Eggli, Y., Van Melle, G., Vagnair, A.: Risk of falls for hospitalized patients: a predictive model based on routinely available data. J. Clin. Epidemiol 54, 66–1258 (2001)
17. Morse, J.M., Prowse, M.D., Morrow, N., Federspeil, G.: A retrospective analysis of patient falls. Can. J. Public Health. 76, 8–116 (1985)
18. Nakai, A., Akeda, M., Kawabata, I.: Incidence and risk factors for inpatient falls in an academic acute-care hospital. J. Nippon Med. School. 73, 70–265 (2006)
19. Schwendimann, R.,Buhler, H.,De Geest, S., Milisen, K.: Falls and consequent injuries in hospitalized patients: effects of an interdisciplinary falls prevention program. BMC Health Serv. Res. 6, 69 (2006)
20. Ash, K.L., MacLeod, P., Clark, L.: A case control study of falls in the hospital setting. J. Gerontol Nurs. 24, 7–15 (1998)
21. Krauss, M.J., Evanoff, B., Hitcho, E., Ngugi, K.E., Dunagan, W.C., Fischer, I., et al: A case-control study of patient, medication, and care-related risk factors for inpatient falls. J. Gen. Intern. Med. 20, 22–116 (2005)
22. Ordem dos Enfermeiros, `http://www.ordemenfermeiros.pt/documentosoficiais/`
23. McGlynn, E.A., Cassel, C.K., Leatherman, S.T., DeChristofaro, A., Smits, H.L.: Establishing national goals for quality improvement. Medical Care. 41, I16—I29 (2003)
24. Griffiths, P., Jones, S., Maben, J., Murrells, T.: State of the Art Metrics for Nursing: A Rapid Appraisal. King's College London, London (2008)
25. Papanikolaoua, P., Lynea, P., Anthony, D.: Risk assessment scales for pressure ulcers. International Journal of Nursing Studies 44, 285—296 (2007)
26. Bennett, G., Dealey, C., Posnett, J.: The Cost of pressure ulcers in the UK. Age and Aging. 33, 230—235 (2004)

Índice de Autores

APOIOS

2ª EDIÇÃO

INFORUM

SIMPÓSIO DE INFORMÁTICA

ISBN
978-989-96863-0-4