Actas do

INForum
Simpósio de Informática
2009

Edição: Faculdade de Ciências da Universidade de Lisboa
Editores: Luís Rodrigues e Rui Lopes
Design Gráfico: Tiago Reis
Fotografia: Dulicon S.A.
Impressão: Serigrafia Varatojo
Ano: 2009
ISBN: 978-972-9348-18-1

Mensagem do Presidente da Comissão Coordenadora

Bem-vindo à primeira edição do INForum, Simpósio de Informática.

Em nome da Comissão Organizadora, é com imensa satisfação que lhe dou as boas-vindas ao INForum, que se estreia este ano na Faculdade de Ciências da Universidade de Lisboa.

O INForum tem como principal objectivo reunir anualmente a comunidade portuguesa de investigação em Informática, proporcionando a sua interacção, a divulgação e discussão de trabalhos, e o contacto com personalidades de reconhecido impacto internacional na área. O INForum é um simpósio multidisciplinar, abrangente e dinâmico com a selecção anual do conjunto de tópicos abordados. Desta forma, é simultaneamente promovida a massa crítica para o fortalecimento e evolução da Informática no país e criado espaço para a emergência de novos interesses de investigação.

O alvo do simpósio é essencialmente nacional, alunos e docentes de cursos de pós-graduação em Informática oferecidos na maioria das instituições de ensino superior nacionais, investigadores das mais de duas dezenas de unidades de investigação reconhecidas pela Fundação para a Ciência e a Tecnologia e de um número crescente de empresas com resultados de I&D de grande relevância internacional. Porém, com a predominância de submissões em língua inglesa é facilitado o reconhecimento do INForum internacionalmente e, uma vez consolidado, será natural esperar o contributo da vasta comunidade de investigadores lusófonos.

O programa técnico do INForum 2009 é rico e diverso no conjunto de problemas abordados. Em números, o evento foi estruturado em 10 tópicos, 48 apresentações, envolveu 30 organizadores, 198 membros de comissões de programa e 149 revisores científicos. O programa conta ainda com 3 ilustres oradores convidados: o Prof. Paulo Veríssimo, da Universidade de Lisboa, que aborda os riscos da segurança de infraestruturas de informação crítica, o Prof. António Câmara, da YDreams, que nos apresenta a sua visão sobre os desafios da média digital e o Dr. Diogo Vasconcelos, da Cisco, que elabora sobre o papel da tecnologia na inovação social.

A elaboração de um programa científico tão diverso e envolvendo tantas pessoas é um processo exigente. Em nome da Comissão Coordenadora, apresento os nossos melhores agradecimentos à empenhada colaboração dos organizadores de cada tópico e à excepcional coordenação dos Prof. Luís Rodrigues e Prof. Rui Pedro Lopes.

O INForum realiza-se durante dois dias intensos, com sessões plenárias e 3 sessões temáticas em paralelo. O programa social inclui um jantar com todos os participantes que serve de momento privilegiado para o contacto e convívio informal. À Comissão Organizadora, Prof. Ana Paula Afonso e Prof. Hugo Mi-

randa agradeço a sua constante disponibilidade e dedicação que, não raramente, ultrapassou as responsabilidades da organização local.

À Comissão para a Imagem e Divulgação, Prof. Dulce Domingos e Prof. Jorge Sousa Pinto, o nosso obrigado pela atempada e abrangente divulgação das várias chamadas à participação no INForum. O nosso agradecimento também à Dra. Bárbara Barroso, ao Eng. Paulo Pombinho, ao Dr. Tiago Grego e ao Eng. Miguel Matos por todo o apoio gráfico e tecnológico dispensado à organização do evento.

Na sua estreia, várias empresas e organizações apoiaram e acompanharam com interesse o INForum: Microsoft, Hewlett-Packard, Critical Software, Banco Espírito Santo, Ordem dos Engenheiros, Large-Scale Informatics Systems Laboratory (FCUL), Centro de Ciências e Tecnologias de Computação (UM), Instituto de Engenharia Electrónica e Telemática de Aveiro (UA) e Center for Informatics and Systems of the University of Coimbra (UC). A todos o nosso muito obrigado.

Rui Oliveira
Comissão Coordenadora

Mensagem da Comissão de Programa

Bem-vindos ao INForum – Simpósio de Informática de 2009. Neste primeiro ano o programa inclui 40 artigos longos e 8 artigos curtos, seleccionados de um conjunto de 88 submissões. Os artigos foram submetidos a um dos tópicos escolhidos pela comissão coordenadora para este ano:

- Compiladores, Linguagens de Programação e Tecnologias Relacionadas (CORTA)
- Computação Distribuída e de Larga Escala
- Computação Móvel e Ubíqua (CMU)
- Encontro de Negócios Digitais
- Engenharia Conduzida por Modelos
- Processamento de Dados, Informação e Linguagem
- Sistemas Inteligentes
- Segurança de Sistemas e Redes de Computadores
- Sistemas Embebidos e de Tempo-Real
- XML: Aplicações e Tecnologias Associadas (XATA)

Cada tópico possui uma comissão científica própria, responsável por recolher a avaliação e seleccionar artigos. Foram aceites submissões dos coordenadores de cada tópico tendo, neste caso, as avaliações sido recolhidas por nós e inseridas de forma anónima. As submissões dos nossos próprios alunos foram também recolhidas pelos coordenadores dos tópicos, garantindo o anonimato.

Atendendo a que vários trabalhos de qualidade não foram seleccionados para apresentação e inclusão nas actas, o INForum inclui uma sessão de "Trabalhos em Curso", que pretende promover a sua divulgação e proporcionar aos seus autores a recolha de sugestões e comentários. Finalmente, o programa conta com a participação de três excelentes oradores convidados.

Alguns dos tópicos do INForum correspondem a eventos já com alguma tradição em Portugal como a CMU, o CORTA ou o XATA. Outros foram organizados de raíz para este evento. Conciliar realidades distintas, com diferentes culturas, tradições e dimensões num único evento é um desafio estimulante. Como coordenadores, definimos o número alvo de avaliações por artigo submetidos (4), tentámos harmonizar os critérios usados em cada tópico e, em função do número e qualidade de submissões em cada tópico, definimos valores de referência para as taxas de aceitação. Graças à excelente cooperação dos vários coordenadores dos diferentes tópicos, a quem publicamente agradecemos, julgamos que se conseguiu um programa com grande coerência, em que o resultado final é muito mais do que a soma das partes.

Antes de acabar, não podemos deixar de agradecer o apoio que sempre recebemos do presidente da comissão coordenadora, Prof. Rui Oliveira, assim como dos membros da organização local, Prof. Ana Afonso e Prof. Hugo Miranda.

Foi uma honra servir o INForum na sua primeira edição,

Luís Rodrigues e Rui Pedro Lopes

Organização

O INForum'09 é organizado pela Faculdade de Ciências da Universidade de Lisboa, onde decorrem as sessões.

Comissões

Comissão de Programa:	Luís Rodrigues (I.S.T.) Rui Pedro Lopes (I.P. Bragança)
Comissão Organizadora:	Ana Paula Afonso (U. Lisboa) Hugo Miranda (U. Lisboa)
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 (I.S.T.) Marco Vieira (U. Coimbra) Rui Lopes (I.P. Bragança) Rui Oliveira (U. Minho), Presidente Simão Sousa (U. Beira Interior)

Comissões de Programa

Compiladores, Linguagens de Programação e Tecnologias Relacionadas (CORTA)

António Leitão (U. T. Lisboa), *chair*	Jorge Sousa Pinto (U. Minho)
Ademar Aguiar (U. Porto)	João José Almeida (U. Minho)
Bastian Cramer (U. Paderborn)	João Costa Seco (U. N. Lisboa)
Boštjan Slivnik (U. Ljubljana)	João Saraiva (U. Minho)
Carlos Fonseca (U. Algarve)	Luís Caires (U. N. Lisboa)
Ivan Lukovic (U. Novi Sad)	Maria João Pereira (I.P. Bragança)
Joost Visser, (SIG)	Mário Florido (U. Porto)

Marjan Mernik (U. Maribor)
Matej Črepinšek (U. Maribor)
Mirjana Ivanovic (U. Novi Sad)
Paulo Matos (I.P. Bragança)
Pedro Guerreiro (U. Algarve)
Pedro Henriques (U. Minho)
Rogério Dias Paulo (Efacec)
Salvador Abreu (U. Évora)
Tomaz Kosar (U. Maribor)
Vasco Vasconcelos (U. Lisboa)
Vitor Santos (Microsoft Portugal)

Computação Distribuída e de Larga Escala

Alysson Bessani (U. Lisboa)
António Sousa (U. Minho)
David Martins de Matos (I.S.T.)
Henrique Domingos (U. N. Lisboa)
Hugo Miranda (U. Lisboa)
João Lourenço (U. N. Lisboa)
João Paulo Carvalho (I.S.T.)
José Leal (Inst. Gulbenkian de Ciência)
José Orlando Pereira (U. Minho)
José Vermelhudo (NAV)
Luís Miguel Pinho (I. S. E. Porto)
Luís Moura e Silva (U. Coimbra)
Luís Oliveira e Silva (I.S.T.)
Luís Veiga (I.S.T.), *chair*
Nuno Duro (Evolve Space Solutions)
Paula Prata (U. Beira Interior)
Paulo Ferreira (I.S.T.)
Paulo Marques (U. Coimbra)
Paulo Vilela (Sun Portugal) Pedro Furtado (U. Coimbra)

Computação Móvel e Ubíqua

Adriano Moreira (U. Minho)
Ana Paula Afonso (U. Lisboa)
André Zúquete (U. Aveiro)
Carlos Baquero (U. Minho)
Carlos Bento (U. Coimbra)
Eduardo Dias (U. N. Lisboa, U. Évora)
Frutuoso Silva (U. Beira Interior)
Joaquim Jorge (I.S.T.)
Lúcio Ferrão (OutSystems)
Luis Veiga (I.S.T.)
Mário Alves (I. S. E. Porto)
Markus Endler (PUC-Rio)
Miguel Monteiro (U. Porto)
Nuno Preguiça (U. N. Lisboa)
Paulo Ferreira (U. T. Lisboa), *chair*
Pedro Araújo (U. Beira Interior)
Renato Cerqueira (PUC-Rio)
Rui José (U. Minho)
Vitor Rodrigues (Movensis)

Processamento de Dados, Informação e Linguagem

Andreas Wichert (I.S.T.)
Bruno Martins (I.S.T.), *chair*
David Matos (I.S.T.)
Francisco Couto (U. Lisboa)
Gaël Dias (U. Beira Interior)
Helena Galhardas (I.S.T.), *chair*
Irene Rodrigues (U. Évora)
João Pereira (I.S.T.)
Luísa Coheur (I.S.T.), *chair*
Maribel Santos (U. Minho)
Orlando Belo (U. Minho) Paulo Carreira (I.S.T.)
Paulo Quaresma (U. Évora)
Pável Calado (I.S.T.), *chair*
Pedro Furtado (U. Coimbra)

Encontro de Negócios Digitais

Alberto Silva (I.S.T.)
Ana Paula Rocha (U. Porto)
António Mesquita (Forum B2B)
António Palma dos Reis (ISEG)
Filomena Lopes (U. Portucalense)
Henrique S. Mamede (U. Aberta)
Isabel Ramos (U. Minho)
J. Dias Coelho (U. N. Lisboa), *chair*
João Varajão (U. Trás-os-Montes e Alto Douro)
Jorge Pereira (Infosistema)
José Adriano Pires (I.P. Bragança)
Leonel Santos (U. Minho)
Luís Amaral (U. Minho), *chair*
Luis Borges Gouveia (U. Fernando Pessoa)
Luís Cabrita (Prológica)
Luís Vidigal (APDSI)
Manuela Cunha (I.P. Cávado Ave)
Paula Morais (U. Portucalense)
Paulo Rita (ISCTE)
Paulo Rupino (U. Coimbra)
Ramiro Gonçalves (U. Trás-os-Montes e Alto Douro)
Rui Quaresma (U. Évora) Vítor Santos (Microsoft Portugal)

Engenharia Conduzida por Modelos

Alberto Silva (I.S.T.), *chair*
Ana Paiva (U. Porto)
António Leitão (I.S.T.)
David Ferreira (I.S.T.)
Fernando Brito de Abreu (U. N. Lisboa)
João Araújo (U. N. Lisboa)
João Pascoal Faria (U. Porto), *chair*
João Paulo Carvalho (Quidgest)
João Saraiva (I.S.T.)
José Borbinha (I.S.T.)
Leonel Nóbrega (U. Madeira)
Levi Lúcio (U. Genebra)
Luís Pedro (U. Genebra) Miguel Calejo (Declarativa)
Nuno Nunes (U. Madeira)
Ricardo Machado (U. Minho)
Vasco Amaral (U. N. Lisboa), *chair*

Sistemas Inteligentes

António Abelha (U. Minho), *chair*
Carlos Ramos (I.P. Porto)
Cesar Analide (U. Minho)
Costin Badica (U. Craiova)
Giuseppe Mangioni (U. Catania)
Goreti Marreiros (I.P. Porto)
João Balsa (U. Lisboa)
José Machado (U. Minho), *chair*
José Neves (U. Minho), *chair*
Luis Moniz (U. Lisboa)
Manuel Santos (U. Minho)
Paulo Cortez (U. Minho)
Paulo Novais (U. Minho), *chair*
Vitor Alves (U. Minho)
Zhaohao Sun (U. Ballarat)

Segurança de Sistemas e Redes de Computadores

Alysson Bessani (U. Lisboa)
André Zúquete (U. Aveiro)
Carlos Ribeiro (I.S.T.)

Edmundo Monteiro (U. Coimbra), *chair*
Henrique Domingos (U. N. Lisboa)
Henrique Madeira (U. Coimbra)
Henrique Santos (U. Minho)
João Barros (U. Porto)
José Alegria (PT)
José Pina Miranda (Multicert)
Luis Antunes (U. Porto)
Manuel Bernardo Barbosa (U. Minho)
Marco Vieira (U. Coimbra)
Miguel Correia (U. Lisboa)
Nuno Neves (U. Lisboa), *chair*
Paulo Ferreira (I.S.T.)
Paulo Simões (U. Coimbra)
Paulo Sousa (U. Lisboa)
Paulo Veríssimo (U. Lisboa)
Pedro Adão (I.S.T.)
Ricardo Chaves (I.S.T.)
Simão Melo Sousa (U. Beira Interior)

Sistemas Embebidos e de Tempo-Real

Adelino Silva (LINCIS)
Carlos Almeida (I.S.T.)
Helder Silva (EDISOFT)
João Almeida (Link Consulting)
João Cardoso (U. Porto)
João Cunha (ISEC)
João Fernandes (U. Minho)
Joaquim Ferreira (U. Lisboa)
José Fonseca (U. Aveiro)
José Malaquias (ISA)
José Metrôlho (I. P. Castelo Branco)
José Rufino (U. Lisboa), *chair*
Leonel Sousa (I.S.T.)
Luís Almeida (U. Porto)
Luís Gomes (U. N. Lisboa)
Luís Miguel Pinho (I. S. E. Porto), *chair*
Mário Calha (U. Lisboa)
Mike Rennie (DEIMOS Engenharia)
Nuno Pereira (I.P. Porto)
Nuno Silva (Critical Software)
Pedro Fonseca (Micro I/O)
Rui Camolino (Brisa)
Simão Sousa (U. Beira Interior)
Tobias Schoofs (Skysoft Portugal)

XML: Aplicações e Tecnologias Associadas (XATA)

Ademar Aguiar (U. Porto)
Alberto Rodrigues da Silva (I.S.T.)
Alberto Simões (U. Minho), *chair*
Alda Lopes Gançarski (INT)
Ana Paula Afonso (U. Lisboa)
Benedita Malheiro (I. S. E. Porto)
Carlos Damásio (U. N. Lisboa)
Cristina Ribeiro (U. Porto)
Daniela da Cruz (U. Minho)
Francisco Couto (U. Lisboa)
Gabriel David (U. Porto)
Giovani Librelotto (UFSM)
João Correia Lopes (U. Porto)
João Moura Pires (U. N. Lisboa)
José Carlos Ramalho (U. Minho)
José João Almeida (U. Minho)
José Luís Borbinha (I.S.T.)
José Paulo Leal (U. Porto)
Luis Carriço (U. Lisboa)
Luis Ferreira (I.P. Cávado Ave)
Marta Jacinto (ITIJ)
Miguel Ferreira (U. Minho)
Nuno Horta (I.S.T.)
Paulo Marques (U. Coimbra)
Pedro Henriques (U. Minho)
Rui Lopes (U. Lisboa)
Salvador Abreu (U. Évora)
Stephane Gançarski (LIP6)
Xavier Gómez Guinovart (U. Vigo)

Comissão de selecção do prémio BES

Cristina Videira Lopes (U. California), Irvine, USA
José Luiz Fiadeiro (U. Leicester), UK
Miguel Castro (Microsoft Research), UK
Pedro Trancoso (U. Cyprus), Cyprus

Revisores Científicos

Adelino Silva
Ademar Aguiar
Adriano Moreira
Alberto Silva
Alberto Simões
Alda Lopes Gançarski
Alysson Bessani
Ana Afonso
Ana Paiva
Ana Rocha
André Zúquete
Andreas Wichert
Antónia Lopes
António Leitão
António Sousa
Bastian Cramer
Benedita Malheiro
Bostjan Slivnik
Bruno Martins
Carlos Almeida
Carlos Baquero
Carlos Damásio
Carlos Fonseca
Carlos Ribeiro
Cristina Ribeiro
Daniela da Cruz
David Ferreira
David Martins de Matos
Diana Santos
Edmundo Monteiro
Eduardo Dias
Fernando Abreu
Francisco Couto
Frutuoso Silva
Gabriel David
Gael Dias
Giovani Librelotto
Helder Silva
Helena Galhardas
Henrique Mamede
Henrique Santos
Hugo Manguinhas
Hugo Miranda
Irene Rodrigues
Ivan Lukovic
Jeferson Souza
João Almeida
João Araújo
João Cardoso
João Carvalho
João Correia Lopes
João Craveiro
João Cunha
João Faria
João Fernandes
João Lourenço
João Moura Pires
João Paulo Carvalho
João Pereira
João Saraiva
João Seco
Joaquim Ferreira
Joost Visser
Jorge Sousa Pinto
José Alegria
José Borbinha
José Carlos Ramalho
José Fonseca
José Malaquias
José Metrôlho
José Paulo Leal
José Pereira
José Pina Miranda
José Rufino
José Vermelhudo
Leonel Sousa
Levi Lúcio
Lúcio Ferrão
Luís Amaral
Luís Antunes
Luís Caires
Luís Carriço
Luís Gomes
Luís Pedro
Luís Pinho
Luís Rodrigues
Luís Silva
Luís Veiga
Luísa Coheur
Manuela Cunha
Marco Vieira
Maria Pereira
Maribel Santos
Mário Alves
Mário Calha
Mário Florido
Marjan Mernik
Markus Endler
Marta Jacinto
Matej Črepinšek
Miguel Correia
Miguel Matos
Miguel Monteiro
Mike Rennie
Mirjana Ivanovic
Nuno Ferreira Neves
Nuno Oliveira
Nuno Pereira

Nuno Preguiça
Nuno Silva
Orlando Belo
Paula Morais
Paula Prata
Paulo Carreira
Paulo Ferreira
Paulo Marques
Paulo Matos
Paulo Quaresma
Paulo Rupino
Paulo Simões
Paulo Sousa
Paulo Veríssimo
Paulo Vilela
Pável Calado
Pedro Adão
Pedro Fonseca
Pedro Furtado
Pedro Guerreiro
Pedro Henriques
Ramiro Gonçalves
Renato Cerqueira
Ricardo Chaves
Ricardo Machado
Rogério Paulo
Rui Camolino
Rui Dinis Sousa
Rui José
Rui Lopes
Rui Oliveira
Rui Quaresma
Salvador Abreu
Simão Sousa
Stephane Gançarski
Tobias Schoofs
Tomaz Kosar
Vasco Amaral
Vasco Vasconcelos
Vitor Santos
Xavier Gómez Guinovart

Apoios

Banco Espírito Santo
Critical Software
Hewlett-Packard
Microsoft

Ordem dos Engenheiros

Center for Informatics and Systems of the Univ. of Coimbra, U. Coimbra
Centro de Ciências e Tecnologias da Computação, U. Minho
Instituto de Engenharia Electrónica e Telemática de Aveiro, U. Aveiro
Large-Scale Informatics Systems Laboratory, U. Lisboa

Índice

Oradores Convidados

Paulo Veríssimo

Paulo Veríssimo é doutorado e agregado em Eng. Electrotécnica e de Computadores, pelo IST. É professor no Departamento de Informática (DI) da Faculdade de Ciências da Universidade de Lisboa e Director do LASIGE, laboratório de investigação do DI. É *Fellow* do IEEE. É editor associado do *Elsevier Int'l Journal on Critical Infrastructure Protection*, foi editor associado das *IEEE Transactions on Dependable and Secure Computing* e pertenceu ao *European Security & Dependability Advisory Board*. Foi Presidente do *IEEE Technical Committee on Fault Tolerant Computing* e do *Steering Committee* da conferência DSN e membro do Conselho Executivo da "CaberNet European Network of Excellence". Foi coordenador do projecto Europeu IST/FET CORTEX. Paulo Veríssimo lidera o Grupo de investigação Navigators integrado no LASIGE e interessa-se correntemente por: arquitectura, algoritmos e subsistemas de suporte (*middleware*) para sistemas distribuídos, embebidos e permeantes (*pervasive*), nas facetas de adaptabilidade em tempo-real e segurança e tolerância a faltas/intrusões. Tem mais de 145 publicações internacionais com revisor, e é co-autor de cinco livros internacionais.

Riscos de Segurança das Infraestruturas de Informação Crítica ou porque Bang! é diferente de Crash

Esta palestra é inspirada numa intervenção recente na revista IEEE Security & Privacy, January 2008, em resposta a uma pergunta no "Information Assurance Technology Forecast 2008":

Sec&Priv: What's the nature and magnitude of risk that critical information infrastructure (CII) faces over the next 15 years? By "critical", I mean the part whose failure would have major effects on the nation, such as economic loss or loss of life.

Paulo Verissimo: Large and ever increasing. Moreover, the objective risk is amplified by the lack of perception of the risk itself existing, by citizens, policy makers, and CII manufacturers and operators. There's still a belief that the SCADA (Supervisory, Control and Data Acquisition) systems controlling these infrastructures are legacy, closed, obscure, and thus unattackable, or that it suffices to just use a firewall and an intrusion detector. But normal ICT systems protection won't be enough. To keep a long story short: Ctl-Alt-Del isn't a remedy for things that have worked continuously for more than 20 years, many security techniques hamper real-time operation, and there's still a difference between erasing a database and setting a generator on fire. This should be understood immediately or else we should get prepared for the next generation of mass hacking. Maybe all it takes for people to get serious about this is a www.scada_rootshell.com (Google the remainders of the classical www.rootshell.com to grasp the basic idea). It might be a good idea for policy makers and CII manufacturers and operators to learn the difference between crash and bang.

Artigo completo na BD do IEEEexplore. Information Assurance Technology Forecast 2008, Steven M. Bellovin, Terry V. Benzel, Bob Blakley, Dorothy E. Denning, Whitfield Diffie, Jeremy Epstein, Paulo Veríssimo. IEEE Security & Privacy, vol. 6, no. 1, pp. 10-17, January/February, 2008.

António Câmara

António Câmara é professor na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. Foi professor convidado na Cornell University (1988-89) e no MIT (1998-99). Esteve ligado ao estudo de impacto ambiental do Alqueva, à reconversão ambiental da Expo'98 e ao Sistema Nacional de Informação Geográfica. Foi fundador Y-Dreams em Junho de 2000, empresa que dirige actualmente.

Challenges in Digital Media

YDreams has developed YVision, a platform for digital media including computer vision, embodied interaction, embodied agents and simulation. The main concepts underlying this platform will be reviewed. YVision addresses "normal" digital media problems. YDreams has been working in three challenging problems that are beyond "normalcy". They include:

- Providing each city driver with an optimal route using information processed from traffic cameras and other sensors;
- Helping a famous soccer coach in optimizing space for his team, both on offense and defense, using video images;
- Translate partial diiferential equations into chemical reactions to enable chemical computing in any surface.

These problems will be defined and our approach to solve them presented. Hopefully, the audience will have better ideas.

Diogo Vasconcelos

Licenciado em Direito, na Universidade Católica do Porto. Pós-graduado em Direito de Comunicação (Universidade de Coimbra) e em Gestão para Licenciados em Direito (Universidade Católica do Porto). Frequentou o mestrado em Ciência Política, na Universidade Católica de Lisboa.

A nível político, Diogo Vasconcelos foi vice-presidente do PSD de Maio de 1999 a Março de 2000 e desde essa altura até às últimas eleições, em Março de 2001 foi o porta-voz do PSD para a área da sociedade da informação. Em Março de 2001 foi eleito deputado à Assembleia da República pelo círculo do Porto e em Outubro do mesmo ano foi nomeado gestor da UMIC – Unidade de Missão Inovação e Conhecimento, a entidade tutelada pelo Ministro Adjunto do Primeiro-Ministro para a sociedade da informação, governo electrónico e inovação. Diogo Vasconcelos é igualmente administrador não executivo da Agência de Inovação e do Instituto Sá Carneiro. Como estudante, foi presidente da Associação de Estudantes da Escola Secundária António Nobre, presidente da Associação de Estudantes da Faculdade de Direito da Universidade Católica, presidente e fundador da Federação Académica do Porto. Representou, eleito em ENDA, os estudantes do Ensino Superior no Conselho Nacional de Educação.

Como empreendedor, foi fundador de várias empresas nas áreas de conteúdos (produção multimédia e revista "Ideias & Negócios"), ambiente e foi co-fundador da primeira capital de risco independente do portuguesa. Deixou as suas actividades profissionais para se dedicar em exclusivo à UMIC.

Foi Vice-presidente da Associação Nacional de Jovens Empresários (1996-2001), no âmbito da qual lançou a Academia dos Empreendedores. Integrou os conselhos consultivos de diversos centros de inovação e incubação de empresas.

Participou, como orador, em dezenas de conferências sobre inovação, venture capital, empreendedorismo e sociedade do conhecimento em Espanha, Reino Unido, Itália, Bélgica, Alemanha, França, Brasil e EUA.

Sessão 1A: Compiladores, Linguagens de Programação e Tecnologias Relacionadas (CORTA)

Bastian Cramer

Bastian Cramer received his degree in Computer Science from the University of Paderborn, Germany in 2005. Then he joined the research group "Programming Languages and Compilers" of Prof. Kastens at the same university. His research focus is the generation of software from specifications and especially the generation of environments for visual domain specific languages. He has several years of experience in language design in corporation with the automotive industry. Currently he is working on his PhD concerning simulation and animation of visual languages.

Simulation and animation of visual languages

Today, visual languages are often used in the software engineering process to specify a system on a high abstraction level. Generator frameworks are often used to generate environments for visual languages. Hence, it is possible to derive even challenging editors for VLs with modest effort and to use them in rapid development processes. Indeed, today's software design process focuses on static visual languages. The simulation and animation of such a language could bridge the gap between program and program execution which must always be in the programmers mind.

This talk presents how we have extended our VL environment generator DEViL (Development Environment for Visual Languages) with simulation and animation support. Simulation is achieved through a domain analysis and the reuse of existing specification concepts in DEViL. An animation is automatically derived from a simulation specification. This results in a formal mapping between simulation and animation. Animation is then a (nearly) linear graphical interpolation which can be adapted by means of 'animation patterns' in a declarative way.

An Exception Aware Behavioral Type System for Object-Oriented Programs

Filipe Militão and Luís Caires

CITI / Departamento de Informática, Universidade Nova de Lisboa, Portugal

We develop a type system for object oriented languages that combines standard type information with behavioral protocol specifications. The typing rules cover familiar constructs, as well as exceptions, which are a main novelty in this work: exceptions may cause abrupt control transfer in allowed behaviors, and have been particularly difficult to tackle with behavioral type systems. The type system guarantees protocol fidelity both at the method level and at the class level by checking consistency in the use of fields with the class' usage protocol. It also ensures that program execution always reaches a safe termination state, even in the presence of behavioral borrowing, that is, temporary aliasing of object references during methods calls.

1 Introduction

The ever increasing software complexity has always been the striving force behind the push for more advanced type systems in an effort to reduce the number of software bugs. Although virtually all modern day verification techniques guarantee the absence of important errors, such as calling non-existing methods or incorrect conversion of data structures, they still leave much room for improvements. Namely, most are incapable of checking the fulfillment of prevalent APIs assumptions such as specific restrictions in the allowed sequences of calls (that is, making sure a protocol is obeyed) or identifying possible sources of interferences in the use of objects (due to aliasing, for instance). Therefore, recent developments in typing methodologies try to address such issues (including other emerging use cases such as web services and concurrency) to protect programs from non-trivial and frequent mistakes.

We focus on the problem of checking the behavior restrictions in the use of methods, that is, how some objects require specific sequences of calls to be observed in order to function properly. File objects are an usual example: they require to be opened before used and closed at the end (to force flushing out any changes). This behavioral protocol is usually defined in attached documentation and/or checked only at run-time (with any error flagged as an exception). We propose a type system that uses a formal behavior description to statically ensure the absence of behavioral errors in sequential programs, so that such protocols are always obeyed from start to finish. While the idea of exploiting behavioral specifications to discipline the usage of objects in programs is not new, in this paper we approach language features much more challenging than in other approaches, such as borrowing behaviors, full class consistency check, and, prominently, exception handling. We have also implemented our type system in a working prototype interpreter and runtime system, the **yak** language [22].

Example 1 shows a $Travel$ class that defines a **usage** protocol specifying its allowed behavior. Then, the **let** expression uses that class accordingly, where each method call

causes a transition in the allowed behavior. The comments show how the behavior of the new object evolves until it reaches a termination state, **stop**. This is but a tiny example: our full protocol description language is capable of describing exceptions, choices, and behavior selection based on the result of a call (as illustrated on Section 3.3).

Example 1. Travel example.

```
class TravelOrder {                    let t in
 usage flight.hotel.(buy+cancel)        t = new TravelOrder();
 void flight(){ ... }                        //TravelOrder#flight.hotel.(buy+cancel)
 void  hotel(){ ... }                   t.flight(); //TravelOrder#hotel.(buy+cancel)
 void    buy(){ ... }                   t.hotel();  //TravelOrder#buy+cancel
 void cancel(){ ... }                   if ( ... ) t.buy() //TravelOrder#stop
}                                       else t.cancel()    //TravelOrder#stop
```

In our language each class declares a behavioral protocol that restricts the use of some of its methods to contexts given by its **usage**. The syntax and semantics (Section 2) is similar to the one of mainstream OO languages. The behavioral information is only used by the type system (Section 3), that handles usual expressions such as branches, loops, exceptions and is also able to check the behavior in cases of recursion, behavioral borrowing and subtyping (Section 3.1). The consistency check procedure (Section 3.2) guarantees behaviors are obeyed, both at the method level and at the class level so that fields are used correctly during the lifetime of an object. Finally, in Section 3.3, we briefly address some flexibility issues of the typing rules and give additional examples. The original contributions of this paper include:

- A simplified type annotation that reduces the burden on the programmer (Section 3), namely, by using a consistency check phase (Section 3.2) that automatically handles the behavior of fields in each method. Additionally, our typing rules do not interfere with type abstraction or modularity and are flexible enough not to require a direct mapping of the protocol into code (Section 3.3). Although due to space limitations we cannot include a formal proof, our type system is provably sound [17].
- Addressing changes in behavior due to raised exceptions has not been much explored even if some proposals [14] explored ideas similar to ours, they have many limitations and are not used in an OO language. In this work, we address behavioral exceptions without compromising much expressiveness (Section 3).
- Although we enforce linear aliasing control [21], we do not require all method arguments to possess unique ownerships. Such features are useful to model usual call-by-reference and is conceptually consistent with borrowing behaviors of object references in limited scopes, such as during method calls (details in Section 3).

2 Syntax and Informal Semantics of the Core Language

We develop our type system for a core Java-like object oriented language, based on ClassicJava [11] (we leave out inheritance, for now). The syntax is shown in Figure 1. A program in this language is formed by several **class** definitions (def) each with a behavioral protocol defined after the **usage** keyword. The entry point is an expression (e), after the list of definitions. In a method call the callee object must be represented by

$$
\begin{array}{rcl}
e & ::= & v \\
& | & e; e \\
& | & x = e \\
& | & \textbf{if}(e)\ e\ \textbf{else}\ e \\
& | & \textbf{while}(e)\ e \\
& | & \textbf{try}\ e\ \textbf{catch}(N\ x)\ e \\
& | & \textbf{throw}\ e \\
& | & \textbf{let}\ x\ \textbf{in}\ e \\
& | & \textbf{new}\ c() \\
& | & \beta.m(e^*)
\end{array}
$$

$$
\begin{array}{rl}
c & \in \text{class names} \\
m & \in \text{method names} \\
x, y, f & \in \text{variable names}
\end{array}
$$

$$
\begin{array}{rcll}
prog & ::= & def^*\ e & \text{(program)} \\
def & ::= & \textbf{class}\ c\ \{ & \text{(class decl)} \\
& & \quad \textbf{usage}\ P & \text{(behavior)} \\
& & \quad field^* & \text{(fields)} \\
& & \quad meth^* & \text{(methods)} \\
& & \} & \\
field & ::= & N\ x & \\
meth & ::= & T\ m(arg^*)\ [\textbf{throws}\ N^*]_{\text{opt}}\ \{e\} & \\
arg & ::= & [\textbf{owned}]_{\text{opt}}\ T\ x & \text{(argument)} \\
v & ::= & \textbf{null} & \\
& | & \textbf{true} \mid \textbf{false} & \\
& | & \beta & \\
\beta & ::= & x \mid \textbf{this} & \\
T, U & ::= & \textbf{void} \mid N\#P & \text{(type)} \\
N & ::= & \textbf{boolean} \mid c & \text{(name)}
\end{array}
$$

Fig. 1. Syntax of the core language.

a variable or a **this** pointer (β). An **owned** modifier informs the type system that an argument requires permissions to be returned or stored in a variable in the method's body (this will be describe in detail in the ownership control section of the type system). Due to lack of space, we cannot fully describe the operational semantics of our language, but this will not hinder much the understanding of the reader since most constructs are standard. Therefore, we move immediately to the most interesting issues in our work, namely the type system.

3 A Behavioral Type System for OO programs

In this section we describe the key ideas behind the design of our type system. We define a *behavioral type* to be a pair of standard (Java-like) type with a behavioral description for that type. Thus, our type notation is of the form:

$$Type \triangleq Name\#Behavior$$

The first part ($Name$) refers to a list of method declarations and to the **usage** protocol (i.e., the initial behavior), both defined in the class declaration for $Name$. The second part ($Behavior$) contains the changing protocol (due to a method call, for instance) that is used during the verification. The language for specifying such usage protocols is defined in (Figure 2). As shown in Figure 1 a type's behavioral protocol is declared after the keyword **usage** in the class definition.

The behavioral descriptions describe the stages of the protocol on which a call to a specific method is allowed. We use a regular expressions-like description language, that includes recursion ($\&r$ to create the recursion point and then r to "jump" back to that position - assuming r to be unique), choice ($+$), allowing us to express alternatives in the protocol, and empty behavior (**stop**). We include a new behavioral construct within

$$
\begin{array}{rll}
P, Q, V, O ::= & m[exception^*].R & \text{(method and continuation)} \\
| & r & \text{(recursion label)} \\
| & \&r(P) & \text{(recursion point)} \\
| & P + P & \text{(choice)} \\
| & \textbf{stop} & \text{(empty)} \\
R ::= & P & \\
| & (\textbf{true}.P) + (\textbf{false}.P) & \text{(result based behavioral choice)} \\
exception ::= & N : P & \text{(behavioral exception, if } N \text{ raised then change to } P)
\end{array}
$$

Fig. 2. Syntax for behavioral descriptions.

method calls ($exception$), to capture exceptional behavior. We also include a construct to express behavioral alternatives, based on the value returned by of a method call (only for booleans, although such a system could be easily extended to support other enumerable types). It is convenient to further explain these two novel constructs :

- "$m[N : P|...].Q$", the description for a method usage. This construction declares that a method (m) can be called on that specific context and that it may throw any of the exceptions inside the square brackets. More precisely, after an exception of type N the allowed behavior for that type changes to P, this is what we call a *behavioral exception*. When no exception is raise the protocol continues with Q;
- "(**true**.P) + (**false**.Q)", declares a change in the allowed behavior based on the result of a **boolean** call. Therefore, this construct is only allowed to appear immediately after a method usage. This description allows the type system to e.g., distinguish between the availability of the $next$ method in an iterator, depending on the value returned by the $hasNext$ method.

The choice construction offers an external decision (where the programmer is free to decide which behavioral path to take); and the exception/result-choice is related to an internal choice as they change the allowed behavior only based on the class' internal code. We categorize the methods of a class in two groups: *behavioral methods* - methods whose name appears in the usage protocol and thus that have their use restricted; and *free* (non-behavioral) methods - methods that do not appear in the protocol and may be freely used in any context regardless of the state of the protocol, for example, methods such as $toString$). When a behavior reaches **stop** only free methods are available. Such categorization of methods remains fixed throughout the life of a type since it is related to the **usage** protocol, not to the dynamic state of a type.

Although our type system is fully static, it needs to track the dynamic state of the objects's protocols. Therefore, the basic typing judgment uses effect-tracking to model such changes, as caused by the evaluation of expressions. It has the following form:

$$\Delta \vdash e : T \rightsquigarrow \Delta'$$

The intuitive reading of such judgment is: the initial environment (Δ) on which the expression (e) is checked to be of type (T) causes some side-effects on the initial environment, and leads to the final environment (Δ'). When expression e is of type **boolean**, Δ' has the form ($\Delta_T|\Delta_F$), reflecting the two possible final states (see [17]).

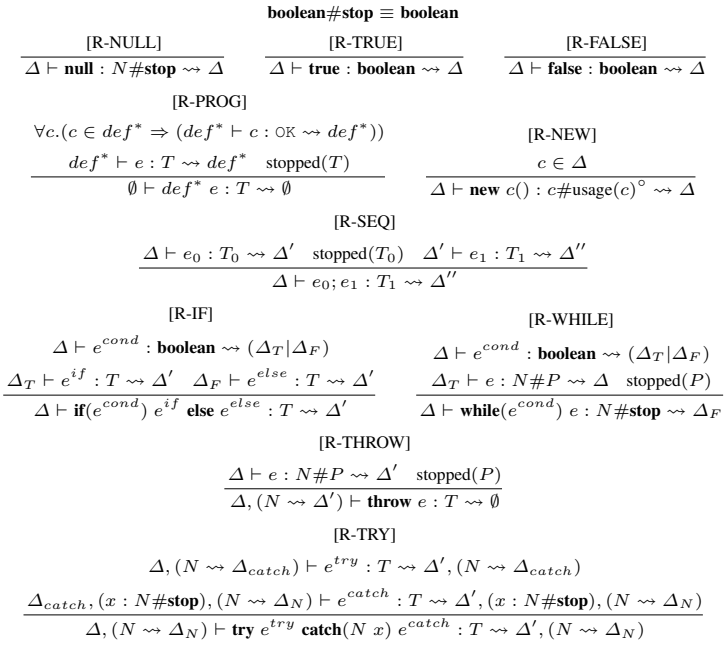


Fig. 3. Basic typing rules.

A typing environment (Δ) is a set of declarations, that may contain:

1. $(x : T)$ - variable (labeled x and of type T);
2. (def) - class definition;
3. $(N \rightsquigarrow \Delta_N)$ - exception handler for a type N where Δ_N is the environment to be used on the nearest **catch** branch of a **try catch** in scope;

Since all the elements of an environment are uniquely declared, we use ($\Delta = \Delta_0, ..., \Delta_n$) to split them into other disjoint sub-environments. In Figure 3 we present the basic typing rules of our type system, we briefly discuss each of them.

R-NULL A **null** value can be used as some type with a **stop** behavior.

R-PROG To check a program we start by checking the consistency of each class (c) before moving to the initial expression (e). Since the resulting value cannot be used elsewhere it must be stopped, the behavior must be able to terminate at this point.

R-NEW The new object starts with the **usage** protocol and is uniquely owned (noted by the $^\circ$) because it is a newly created value.

R-SEQ Since it ignores the result of its left side (T_0), we require that type to be stopped so no behavior is lost. Any side-effects that it may produce are carried on to the right side by the environment Δ' which produces the final result and the resulting environment of this expression.

R-IF Follows the usual **if else** flow with a double environment ($\Delta_T|\Delta_F$) for the case when there is a result-based behavioral choice in e^{cond}.

$$\frac{\Delta, (x : N\#\mathbf{stop}^\circ) \vdash e : T \rightsquigarrow \Delta', (x : N\#P^\circ) \quad \text{stopped}(P)}{\Delta \vdash \mathbf{let}\ x\ \mathbf{in}\ e : T \rightsquigarrow \Delta'}\ [\text{R-LET}]$$

$$\frac{\Delta \vdash e : N\#P^\circ \rightsquigarrow \Delta', (x : N\#Q^\circ) \quad \text{stopped}(Q)}{\Delta \vdash x = e : N\#\mathbf{stop} \rightsquigarrow \Delta', (x : N\#P^\circ)}\ [\text{R-ASSIGN}]$$

$$\frac{P \xrightarrow{P} \mathbf{stop}}{\Delta, (\beta : N\#P^\circ) \vdash \beta : N\#P^\circ \rightsquigarrow \Delta, (\beta : N\#\mathbf{stop}^\circ)}\ [\text{R-TAKE}]$$

$$\frac{P \xrightarrow{V} Q}{\Delta, (\beta : N\#P) \vdash \beta : N\#V^\bullet \rightsquigarrow \Delta, (\beta : N\#Q)}\ [\text{R-BORROW}]$$

Fig. 4. Ownership control typing rules.

R-WHILE The Δ environment models the loop invariant. The result type ($N\#\mathbf{stop}$) ignores the behavior P since we defined the **while** body as leaving a **null** result.

R-THROW The run-time **catch** mechanism does not handle behavior on the raised object, thus its behavior (P) must be stopped. The exception handler defines that the catch environment must be the same as the one resulting from e. The empty environment ($\emptyset$) signals that any following expression will be unreachable.

R-TRY The catch environment (Δ_{catch}) for the type N is used as a handler inside the **try** branch and as the initial environment in the **catch**. Both branches produce the same environment (Δ') so that the behavior afterwards is independent of what happens at run-time, note the previous handler for type N is restored.

Any type system as ours, that tracks the flow of calls, has to deal with another important issue: aliasing control. Aliasing induces interference, which may easily break the prescribed usage protocols. We decided to use a kind of behavioral linearity for controlling access to each object's behavior. Thus, there is no true aliasing as the full behavior is only visible to one variable at a time. However, the non-behavioral or free view of any object has no such restrictions since the use of a stopped type never causes behavioral interferences. Nonetheless, using only this limited view on aliasing would be too restrictive and so we decided to include the option of "borrowing" these behaviors for arguments in calls. Thus, an object can be lent for the specific duration of a call and afterwards continue to be used as if it were always owned by the original variable.

To model this phenomenon, we introduce a simulation operation ($P \xrightarrow{Q} V$). We assert $P \xrightarrow{Q} V$ when an object subject to the usage protocol P, may still be used as defined by V, after a temporary usage as specified by Q. The simulation relation is formally defined by a set of rules, that we cannot include here for lack of space (see [17]). The simulation relation also deals with exceptions, since the argument's protocol Q must be fulfilled in all situations, even when an exception is raised. This solution leads to a whole new set of problems that we need to solve, namely: how can we save a value in a field or return it if it could be borrowed by someone else? For this reason, we use the concept of ownership to distinguish when an object is or not uniquely owned or if that uniqueness is required in a call. Thus, it is only possible to return values or save them in fields if they are **owned** (noted T°). After an ownership is taken away, the variable only retains a stopped view of that object. In the case of borrowed behaviors, the variable does not truly own the object, instead it has a non-owned type (noted $T^\bullet$) that cannot be returned or assigned (thus, is incompatible with the owned type) but still can call the

$$\text{[R-CALLL]}$$

$$\frac{\begin{array}{c} m(T_i)[N_j]\ T^\circ \in \text{methods}(N) \quad i \in \{0,...,n\} \quad j \in \{0,...,k\} \\ \Delta = \bigcup_{i=0}^{n} \Delta_{e_i} \quad \Delta_{e_i} \vdash e_i : T_i \rightsquigarrow \Delta'_{e_i} \quad \bigcup_{i=0}^{n} \Delta'_{e_i} = \Delta', \Delta_{f-before}, (N_j \rightsquigarrow \Delta_{catch_j}) \\ \beta \overset{m}{\rightsquigarrow} O|P \quad \Delta_{f-before} \overset{\beta.m}{\rightsquigarrow} \Delta_{f-after} \qquad \beta \overset{m.N_j}{\rightsquigarrow} V_j \quad \Delta_{f-before} \overset{\beta.m.N_j}{\rightsquigarrow} \Delta_{f-catch_j} \\ \Delta''_O = \Delta', \Delta_{f-after}, (\beta : N\#O) \qquad \Delta_{catch_j} = \Delta', \Delta_{f-catch_j}, (\beta : N\#V_j) \\ \Delta''_P = \Delta', \Delta_{f-after}, (\beta : N\#P) \end{array}}{\Delta, (\beta : N\#Q) \vdash \beta.m(e_0,...,e_n) : T^\circ \rightsquigarrow \Delta''_O|\Delta''_P}$$

Fig. 5. Method Call typing rule.

methods of its protocol. When no explicit ownership notation (T) is given it is assumed that any ownership value will do. We now explain some key aspects of the ownership control typing rules.

R-LET Creates a variable (x, with an initial type compatible with **null**) to be used inside its body (e). Therefore, before it falls out of scope we must make sure that any remaining behavior it may have ($N\#P$) is stopped.

R-ASSIGN An assignment can only occur with variables that are owned. Thus, the old content ($N\#Q$) will be lost and must be stoppable.

R-TAKE Taking a content (owned read) is removing all the behavior it may have. It will lose the possession of any behavior, keeping only a **stop** state.

R-BORROW A borrow read causes some of the behavior to be read in a non-owned way. That is, it cuts a prefix of the behavior and what remains can continue to be used afterwards. This kind of read can only occur when checking an argument of a method call and as such, together with the disjoint checking of arguments (that will be explained further down), it makes it possible to do behavioral borrowing by using those types for non-owned arguments.

Example 2. Some examples involving ownership control.

```
let v in
 v = new C();
//v: C#a.b.c
//"void m(C#a.b x)"
//'borrows' C#a.b
 m(v); //v: C#c
 v.c() //v: C#stop
```

```
let v in
  v = new F();
//v: F#a.(b+stop)
//"void m(owned F#a x)"
// => 'x' requires ownership
// F#a.(b+stop) <: F#a
  m(v); //#b+stop  is hidden
//v: F#stop
```

```
let v0 in
 let v1 in
  v0 = new D();
//v0: D#a     v1: D#stop
  v1 = v0;
//v0: D#stop  v1: D#a
  v1.a()
//v0: D#stop  v1: D#stop
```

R-CALL This is the most complex rule of out type system. First, each argument is checked in a disjoint sub-environment (Δ_{e_i}) that excludes any interferences between them. Then, it must move the behavior of the caller from state Q to the normal (non-exceptional) behavior after the m call (written $O|P$: an alternative behavior to account for result-based behavioral choices; equal alternatives are used if it there is no such choice). This effect is captured by the assertion $\beta \overset{m}{\rightsquigarrow} O|P$, which is defined from the simulation relation (see [17]). Self-inflicted calls do not cause changes in the behavior: from our point of view, the protocol is only meant to express restrictions to clients of

the object (i.e. from the outside) and not internally. Thus, an object may freely call any of its own methods without causing a change in its current behavior. However, the type system must anyway keep track of changes in the behavior of the object fields, as stated in the $\Delta_{f-after}$ environment with the field changes caused the $\beta.m$ call. The type rule also needs to take into account the possibility of the method call raising an exception. To take care of that, a verification similar to the one expressed in the R-THROW rule must be performed for each raisable exception (N_j). However, we must account for behavioral exceptions by stepping the state to the exception behavior V_j while also considering possible changes in the fields ($\Delta_{f-catch_j}$).

3.1 Subtyping

In order to improve flexibility, our type system also includes a rich subtyping relation. The most interesting feature of our subtyping by structure is the use of behavioral protocol compatibility. Thus, in our setting, subtyping must also take into account the interchangeability of usage protocols (so that protocols can be safely replaced without violating expectations) which is achieved directly with the simulation operation.

As described above, we split a class' methods into two groups: behavioral and free methods. A subtype cannot move a method between these two groups, thus a behavioral method in a subtype T must also be behavioral in the supertype U (and an identical situation with free methods). In general, a type T is a subtype of a type U ($T <: U$) if:

- for each method in U, there must exist a method in T with the same name and with a compatible method signature (usual method subtype) while also belonging to the same behavioral/free group. As usual, the subtype is free to define additional methods in any group.
- the simulation of U's current behavior protocol must be compatible with the protocol of T, that is, by simulating in T the protocol of U it must be able to reach a **stop** state so that the subtype includes at least all the behavior of the supertype.

Example 3.

$$TravelOrder\#hotel.(buy + cancel) \qquad Order\#(buy + cancel)$$

These two types are incompatible: the $TravelOrder$ requires a method "$hotel$" that does not exist in $Order$ and both protocols are incompatible. However, once $TravelOrder$ calls "$hotel$" the following relation becomes valid (but not the reversed):

$$TravelOrder\#(buy + cancel) <: Order\#(buy + cancel)$$

Notice that the condition for the behavioral/free methods is based on the **usage** protocol of each class. However, the protocol compatibility (through the simulation operation) uses the current state of the behavior and therefore this relation now holds.

3.2 Consistency Check

Another important feature of our type system is that it verifies that client code respects usage protocols, but also that server code (e.g., classes) implement the usage protocols they declare. Although we will not go into formal details, this is achieved by a

consistency check of the use of fields throughout a class' behavior (testing if it is OK). Essentially, it carries the behavior of fields over all possible paths of the **usage** protocol. Thus, these variables start with a **stop** behavior and at each termination point of the protocol they must also be in a stoppable state. For free methods, their use of fields is restricted to a constant and stopped state so that they cannot interfere with behavioral methods. Finally, at each methods, we must also check that all arguments have been completely used. We illustrate this consistency checking with a simple example.

Example 4. An example of *consistency check* with recursion.

```
class C {              // behavior paths: a -> b
   usage a.(b+c)       //                   -> c
   N v;                //
   void a(){           // a << [v: N#stop]
      v = new N()      //   v : N#m1+m2
   }                   // a >> [v: N#m1+m2]
   void b(){           // b << [v: N#m1+m2]
      if( ... )        //  -> v: N#m1+m2  (no change)
        ( v.m2();      //   v: N#stop
          v = new N(); //   v: N#m1+m2
          this.b() )   //   { v: N#m1+m2 } -(b)-> { v: N#stop }
      else v.m1()      //   v: N#stop
   }                   // b >> [v: N#stop]
   void c(){           // c << [v: N#m1+m2]
      this.b()         //   { v: N#m1+m2 } -(b)-> { v: N#stop }
   } }                 // c >> [v: N#stop]
```

3.3 Discussion and Further Examples

So far, some of the presented typing rules may seem too restrictive as, for example, they require exact matches in the environments of different branches. For this reason, we will briefly discuss some improvements to the basic type rules, that offer additional flexibility. Essentially, we define more flexible ways of combining typing environments.

Example 5. The *environment subtyping* allows for an environment to be safely used as another if they have compatible content and any "extra" variables are stopped.

$$\{(x : N\#b + d), (w : N\#u[M : q]), (y : N\#\mathbf{stop})\} <: \{(x : N\#d), (w : N\#u[M : q|N : w])\}$$

Example 6. The *environment intersection* merges two environments into one that contains the common behavior (and therefore, is valid in both of the initial environments) and any name that does no appear in both will be added to the final environment.

$$\{(x : N\#b + d), (y : N\#\mathbf{stop})\} \sqcap \{(x : N\#q + d)\} = \{(x : N\#d), (y : N\#\mathbf{stop})\}$$

These relations allow us to define how a double environment can be converted to a single one ($(\Delta_T|\Delta_F) <: (\Delta_T \sqcap \Delta_F)$) and conversely that any environment is a double environment where both alternatives are itself ($\Delta = (\Delta|\Delta)$). Therefore, the rules can adapt to cases where a double environment is not produced or when one is not required. By combining these operations with the typing rules we gain some additional flexibility. For instance, the R-IF may have different environments in its branches that are merged using environment intersection. Or the R-THROW rule does not need to produce exactly the same environment as the one in the exception handler, it just need to be an environment subtype of it. In conclusion, different environments can be merged using intersection and two environments are compatible if the subtype relation holds.

We leave the presentation of our formal proof of soundness of the type system, based on subject-reduction and type-safety proofs, to the companion technical report [17]. Before concluding the section, we present some examples of programs and their typings.

Example 7. Consistency check with behavioral exceptions.

```
class C {                    // paths: a -> b -> c
   usage a.b[boolean: c].c   //               -> throw boolean -> c
   N v;                      //
   void a(){                 // a << [v: N#stop]
      v = new N()            //   v : N#(b.(b+c))+(d.(d+c))
   }                         // a >> [v: N#(b.(b+c))+(d.(d+c))]
   void b() throws boolean{  // b << [v: N#(b.(b+c))+(d.(d+c))]
      if( ... )              //
         ( v.d();            //   v: N#d+c
         throw true )        //   throw boolean >> [v: N#d+c]
      else v.b()             //   v: N#b+c
   }                         // b >> [v: N#b+c] (by 'v: N#b+c' intrs '{empty}')
   void c(){                 // c << [v: N#d+c]    c << [v: N#b+c]
      v.c()                  //   v : N#stop
   } }                       // c >> [v: N#stop]   c >> [v: N#stop]
```

Example 8. The **while** body makes a choice that will have behavioral repercussions in the following cycles. Thus, the initial environment cannot be used directly, it must first be subtyped into one that correctly models the loop invariant, $(v : N\#\&r(b.r + stop))$. Since we cannot statically know how many times it will loop (or if it will at all) using a call to "a" after that **while** is always considered to be illegal since the while's body made a behavioral choice and thus changed the allowed behavior.

```
void method(N#a+&r(b.r+stop) v){ // -> v: N#a+&r(b.r+stop)
  while( ... )                   // subtyping environment to
     v.b()                       // v: N#&r(b.r+stop)
}                                // <- v: N#&r(b.r+stop) (stoppable)
```

Example 9. The environment intersection can be used to merge two distinct branches (**if** and **else**) with different behaviors into an environment that contains the shared protocol.

```
void method(N#a+b+stop v){ // -> v: N#a+b+stop
 if( ... ) v.b()           // v: N#stop
 else v.toString()         // v: N#a+b+stop  (toString is 'free')
}                          // <- intersection results in 'v: N#stop'
```

Example 10. Result based choice and exceptions.

```
void m( N#a.(true.b+false.c) v ){ // >> [ v : N#a.(true.b+false.c) ]
   try ( if( v.a() )              // v: N#b | v: N#c
           throw true             //  --> throw boolean + v: N#b
         else v.c()               // v: N#stop
       )                          // v: N#stop ( 'v: N#stop' intrs '{empty}' )
   catch(boolean b)               //  <-- catch boolean + v: N#b
     v.b()                        // v: N#stop
}                                 // << [ v: N#stop ]
```

4 Related Work

The core idea behind our types is based on the spatial-behavioral type system of [6], where it is developed a type system for a resource aware model of behavior, using the π-calculus as the underlying model. This work is an attempt to adapt and expand some of his ideas to a mainstream Java-like language and was developed during Militão's Masters thesis [16], that also lead to a publicly available prototype [22]. This line of research has its remote roots in Nierstrasz's regular types for objects [18].

DeLine and Fähdrich [8,7,9] explore the idea of enforcing protocols in an object oriented language using pre/post conditions to check invariants, that can include a state-machine like protocol. This work also includes features similar to those of ESC/Java [10], by checking other kinds of properties. They includes rules for inheritance (which we do not handle) and subtyping based on a simplified version of behavioral subtyping as proposed by Liskov, et al [15]; even they still allows substitutability violations in some cases. Unlike in our work, they do not use linear typing to ensure sound state transitions, they do not tackle with exception handling, neither consider behavioral borrowing on references passed to method calls. Typestates [19] for objects have been further developed by Aldrich and Bierhoff [2,3,4]. where subtyping relation, by means of state refinement respecting substitutability is defined. Building on the notion of fractional permissions [5], they define *access permissions* [3] whose expressiveness goes beyond linear types. These allow for advanced aliasing control, for example, it is even capable of modeling some situations where it can statically verify the absence of concurrent modifications in the use of iterators [1]. Nonetheless, they do not consider behavioral borrowing as we do here.

A different approach [20] adapts session-types [12] to define dynamic interfaces, that controls access to object methods. This work differs from ours mainly in that they do not guarantee termination of behaviors, nor account for exceptions. The aliasing control is similar, as they also require linearity but without the option of borrowing.

In [13] Igarashi and Kobayashi create a type system for a call-by-value, simply-typed λ-calculus based language that guarantees usage correctness. In later work [14], they expand their proposal to include exceptions similar to our behavioral exceptions. However, they limit the raise construct to a single typeless exception at a time, thus it is not possible to jump to a specific catch branch based on the type of the thrown object, as we do, and is often needed in realistic programs.

5 Concluding Remarks

We have presented a behavioral type system for object oriented programs that statically verifies usage conformance of objects by enforcing behavioral fidelity and termination of protocols declared in class specifications, while extending other existing proposals with flexible aliasing control, and exception handling. We implemented a version of this type system into a prototype interpreter [22], and we have provided a formal proof of its correctness [17]. Our use of a consistency check phase reduces the burden on the programmer by automatically checking the correct use of object fields in accordance to the declared behavioral protocol, without the need for additional annotations. All constructions (exceptions, branches and loops) are checked in a flexible way, that does not require the usage protocol to be directly expressed in the client code. A subtyping relation guarantees that all behavioral expectations are met in accordance with the general substitutability principle. Although we use a linear ownership control with the notion of owned and non-owned types, we account for the possibility of borrowing types in well defined scopes and in a coherent way with normal call-by-reference. We hope in the future to be able to extend our verification techniques to programs with concurrency.

References

1. Kevin Bierhoff. Iterator specification with typestates. In *SAVCBS '06: Proceedings of the 2006 conference on Specification and verification of component-based systems*, pages 79–82, New York, NY, USA, 2006. ACM.
2. Kevin Bierhoff and Jonathan Aldrich. Lightweight object specification with typestates. In Michel Wermelinger and Harald Gall, editors, *ESEC/SIGSOFT FSE*, pages 217–226. ACM, 2005.
3. Kevin Bierhoff and Jonathan Aldrich. Modular typestate checking of aliased objects. In Richard P. Gabriel, David F. Bacon, Cristina Videira Lopes, and Guy L. Steele Jr., editors, *OOPSLA*, pages 301–320. ACM, 2007.
4. Kevin Bierhoff and Jonathan Aldrich. Plural: checking protocol compliance under aliasing. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *ICSE Companion*, pages 971–972. ACM, 2008.
5. John Boyland. Checking interference with fractional permissions. In Radhia Cousot, editor, *SAS*, volume 2694 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2003.
6. Luís Caires. Spatial-behavioral types for concurrency and resource control in distributed systems. *Theor. Comput. Sci.*, 402(2-3):120–141, 2008.
7. R. DeLine and M. Fahndrich. The fugue protocol checker: Is your software baroque, 2003.
8. Robert DeLine and Manuel Fähndrich. Enforcing high-level protocols in low-level software. In *PLDI*, pages 59–69, 2001.
9. Robert DeLine and Manuel Fähndrich. Typestates for objects. In Martin Odersky, editor, *ECOOP*, volume 3086 of *Lecture Notes in Computer Science*, pages 465–490. Springer, 2004.
10. Cormac Flanagan, K. Rustan M. Leino, Mark Lillibridge, Greg Nelson, James B. Saxe, and Raymie Stata. Extended static checking for java. In *PLDI*, pages 234–245, 2002.
11. Matthew Flatt, Shriram Krishnamurthi, and Matthias Felleisen. Classes and mixins. In *POPL*, pages 171–183, 1998.
12. Simon J. Gay and Malcolm Hole. Types and subtypes for client-server interactions. In S. Doaitse Swierstra, editor, *ESOP*, volume 1576 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 1999.
13. Atsushi Igarashi and Naoki Kobayashi. Resource usage analysis. In *POPL*, pages 331–342, 2002.
14. Futoshi Iwama, Atsushi Igarashi, and Naoki Kobayashi. Resource usage analysis for a functional language with exceptions. In John Hatcliff and Frank Tip, editors, *PEPM*, pages 38–47. ACM, 2006.
15. Barbara Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994.
16. Filipe Militão. Design and implementation of a behaviorally typed programming system for web services. Master's thesis, Universidade Nova de Lisboa, July 2008.
17. Filipe Militão and Luís Caires. An exception aware behavioral type system for object-oriented programs. Technical Report UNL-DI-3-2009, CITI / FCT-UNL, 2009.
18. Oscar Nierstrasz. Regular types for active objects. In *OOPSLA*, pages 1–15, 1993.
19. Robert E. Strom and Shaula Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Trans. Software Eng.*, 12(1):157–171, 1986.
20. Vasco T. Vasconcelos, Simon Gay, António Ravara, Nils Gesbert, and Alexandre Z. Caldeira. Dynamic interfaces. In *International Workshop on Foundations of Object-Oriented Languages (FOOL'09)*, 2009.
21. Philip Wadler. Linear types can change the world! In *Programming Concepts and Methods*. North, 1990.
22. yak home page. `http://ctp.di.fct.unl.pt/yak/`.

JaSPEx: Speculative Parallel Execution of Java Applications*

Ivo Anjo and João Cachopo

ESW
INESC-ID Lisboa/Instituto Superior Técnico/Universidade Técnica de Lisboa
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{ivo.anjo,joao.cachopo}@ist.utl.pt

Abstract. Multicore processors, capable of running multiple hardware threads concurrently, are becoming common on servers, desktops, laptops, and even smaller systems. Unfortunately, most of the time these new machines are underutilized, as most current software is not written to take advantage of multiple processors. Also, with these new machines, more cores do not translate into more sequential performance, and existing sequential applications will not speed up by moving to a multicore. To tackle this problem, we propose to use thread-level speculation based on a Software Transactional Memory to parallelize automatically sequential programs. We describe the JaSPEx system, which is able to do automatic parallelization of existing sequential programs that execute on the Java Virtual Machine, and we address the problem of transactifying an existing program and the difficulties inherent to this process. Besides the transactification process, we describe how speculation is introduced and controlled by the JaSPEx system, and what is the relationship between the speculative execution of a program and the Software Transactional Memory that it is using.

Key words: Thread-level Speculation, Transactional Memory, Legacy Applications, Multicore Architectures

1 Introduction

The transition to multicore architectures is ongoing. Chip designers are no longer racing to design the fastest uniprocessor, instead turning to parallel architectures, capable of running many threads simultaneously.

The full power of these multicore chips is unlocked only when all cores are busy executing code. Yet, most desktop applications fail to take advantage of these processors, having little, if any, parallelism. This means that upgrading to a newer processor with more processing cores does not benefit these applications.

Moreover, even if newly developed applications are written with multicore architectures in mind, most of the already developed code is still sequential and

* This work was partially supported by the Pastramy project (PTDC/EIA/72405/2006).

it is not feasible to rewrite it within a reasonable time frame. Thus, an enticing alternative is to parallelize applications automatically. In fact, there is already significant research towards this goal.

For instance, parallelizing compilers [1,2] try to automatically extract concurrency from a sequential program description, while still maintaining program correctness. The problem is that they still fail to parallelize many applications, because of data and interprocedural dependencies that are very hard to analyze at compile-time in a fully static way.

This work explores a different approach – speculative parallelization. Rather than parallelizing only code that is provably able to run in parallel, speculative parallelization uses a more aggressive approach that parallelizes code that may have dependencies, and relies on the ability to roll back a speculative execution when it detects that the parallelization could not have been done.

Unlike other approaches to automatic parallelization that rely on hardware-supported speculative execution (e.g., [3,4,5]), the distinguishing feature of our proposal is the use of a software transactional memory (STM) [6,7] to back up the speculative execution. To the best of our knowledge, we are the first to propose the use of an STM for speculative parallelization.

We argue that using an STM for speculative execution has several advantages over hardware-supported approaches. First, because STM-based executions are unbounded, we may extend the range of possible speculative parallelizations, thereby increasing the potential for extracting parallelism from sequential applications. Second, we may apply these techniques to applications that run on hardware that does not support speculative execution (including all of the current mainstream hardware). Finally, we may leverage on much of the intense research being done in the area of transactional memory.

Yet, switching from hardware-supported speculation to an STM-based approach, introduces other challenges, such as being able to transactify a program to run it speculatively. In this paper, we describe JaSPEx – the Java Speculative Parallel Executor – a system that automatically parallelizes programs for the Java Virtual Machine (JVM) using an STM-based speculative approach. JaSPEx rewrites the bytecode as it is loaded by the JVM runtime, modifying it to run speculatively on top of an STM.

The remainder of this work is organized as follows. Section 2 introduces problems and solutions found for running code speculatively, and further describes the implementation of JaSPEx. Section 3 presents experimental results. Section 4 presents important research related to this work, and, finally, Section 5 summarizes the current findings and future work.

2 Design and implementation

We may parallelize the execution of a Java method like the one shown in Figure 1 by executing the calls to `doA` and `doB` in parallel. The problem is, these methods might modify and access some shared state, and as such may not be able to run in parallel.

```
void method() {
    doA();
    doB();
}
```

Fig. 1. Example method to be parallelized.

Using a speculative approach to the parallelization of programs entails having the ability to detect when a speculative execution violates sequential execution semantics, and the ability to reverse the changes done by a speculative execution when such a violation occurs.

JaSPEx consists of two main elements: (1) a static modification module that acts as a Java class loader, transforming and preparing classes as they are requested by the application; and (2) a runtime control module that performs the speculative executions, coordinating the start, end, termination and return of values from these executions.

The static modification module applies the transformations at load-time, via Java bytecode rewriting, using the ASM bytecode manipulation framework [8]. Sections 2.1, 2.2, and 2.3 describe these transformations. But, because looking into the transformations made at the bytecode level is harder, in this paper we present the transformations as semantically equivalent changes at the Java programming language level.

The runtime control module relies on the changes made by the static modification module, and is responsible for all runtime decisions and control regarding speculation. It is described in Section 2.4.

2.1 Transactification of an application

Because the JVM runtime has no support for transactional execution of code, an application must first be modified to run transactionally, so that the automatic parallelization system is able to detect when a speculative execution violates sequential execution semantics, and is able to reverse the changes made by a speculative execution when such a violation occurs.

To solve this problem, we propose the use of a software transactional memory [6,7] to allow (parts of) the program memory to act transactionally. Execution of different parts of the application is then mapped to different transactions each executing on their own thread, and when there is a conflict between two transactions we know that there has been a violation of sequential execution semantics, and abort the one that comes later in the original program execution.

Coming back to the example in Figure 1, we can parallelize execution of `method` by running `doA` and `doB` in separate threads, each with a different transaction. If the STM system detects a conflict between the speculative execution of `doA` and `doB`, we abort `doB`, and schedule it for reexecution, because the original program order puts `doA` before `doB`; if no conflict is detected, the two meth-

ods are run in parallel, and this should result in a speedup over the sequential version.

The software transactional memory currently used for JaSPEx is the Java Versioned Software Transactional Memory (JVSTM) [9,10], which is a pure Java STM that introduces the concept of versioned boxes, which are containers that keep the history of the values of an object, each of these corresponding to a change made to the box by a committed transaction. The JVSTM was chosen for its features and due to our familiarity with it, but our approach can also be used with other STMs.

As a Java library, applications have to explicitly call the JVSTM to start and end transactions, and Java classes have to be modified to hold `jvstm.VBox` instances, instead of instances of the original object types, as shown in Figures 2 and 3. This process, which we call *transactification* of a class, has to be applied to all classes of a target application, so that it runs entirely under the control of the JVSTM.

```
public class A {
    private String s;

    public A(String s) { this.s = s; }

    public String s() { return s; }
}
```

Fig. 2. Original `A` class.

```
public class A {
    private VBox<String> $box_s = new VBox<String>();

    public A(String s) { $box_s_put(s); }

    public String s() { return $box_s_get(); }

    private String $box_s_get() { return $box_s.get(); }
    private void $box_s_put(String s) { $box_s.put(s); }
}
```

Fig. 3. Transactified `A` class.

The transactification process does the following:

- Replaces the original fields of each class with `private VBox<OriginalType>` fields named `$box_FieldName`.
- Creates the *get* and *put* methods, `$box_FieldName_get` and `$box_FieldName_put`, which mediate access to the corresponding `VBox`. These methods have the same access level as the original field.

– Adds `VBox` slot initializations to the class constructors.
– Replaces accesses to the original fields, either from the same class or from outside classes, with calls to the *get* and *put* methods.

Unfortunately, not all things can be transactified. For instance, `native` methods cannot be analyzed or modified easily. Also, the Sun JVM reserves the `java.*` package namespace and does not allow loading at runtime modified versions of classes within this package or any of its subpackages. We refer to a class that cannot or should not be modified as an *unmodifiable* class.[1]

Besides these unmodifiable classes, there are other features of the Java language and runtime that make the transactification process harder. Arrays cause a multitude of problems. Not only because individual array positions have to be transactified, but specially because transactifying them causes changes to the API of transactified classes, as arrays of a given `OriginalType` have to be replaced by arrays of `VBox<OriginalType>`.[2] This means that all method signatures receiving or returning arrays have to be changed to accommodate this change, which, as we stated before, is not possible on the Sun JVM. Another source of problems is the use of reflection, because it eludes the static transformation of accesses to fields. So, reflection has to be forbidden during speculation, or else modified to be speculation aware. Finally, I/O operations generally cannot be undone.

Because not all things can be transactified, our system must be able to detect all of these cases and make sure such invocations are forbidden during speculative execution.

2.2 Prevention of nontransactional operations

There are two main approaches to prevent the execution of nontransactional operations within a speculative execution: (1) static identification of these operations; and (2) dynamic, runtime prevention of their execution. Static identification consists of building a graph of possible method invocations: If method `A` may call `native` method `B`, then both `A` and `B` are marked as nontransactional. Note that, even though there may be a control flow from `A` to `B`, it does not mean that `A` calls `B` each time it executes. So, as this approach is very conservative, we opted for a dynamic runtime scheme, where methods are modified to invoke the speculation system to check if they can perform nontransactional operations. This way, we can take advantage of the fact that `A` might not invoke `B` very often, and delegate the decision of whether to speculate the execution of `A` for runtime.

JaSPEx supports two modes of code execution: (1) transactified execution, where code is run transactionally but no speculation occurs; and (2) a speculative execution mode, where code runs both transactionally and speculatively.

[1] Including *should not* in this definition is useful because there are other classes that we do not want to modify, such as the `jvstm` libraries, and parts of the JaSPEx framework.

[2] This change can cause further problems, because generics in Java are implemented using *type erasure* [11].

To support speculative execution, JaSPEx creates a speculative version of each method `M`, called `M$speculative`, except for constructors, which always have to be named `<init>`. In this latter case, an alternative scheme is used: A new parameter of type `SpeculativeCtorMarker` is added at the end of every speculative constructor. The speculative version of each method is a copy of the original method with invocations to other methods replaced by calls to their `$speculative` versions, if possible; for nontransactional method invocations, nontransactional field accesses,[3] and operations involving arrays, it adds an invocation to the JaSPEx runtime before performing the operation, so that the speculation system can decide how to proceed.

Additionally, if the original method was native, its `$speculative` counterpart consists of a call to the JaSPEx runtime, followed by a call to the original version. Similarly, because a class may inherit methods from an unmodifiable class, it needs to add `$speculative` versions of inherited methods that call the runtime and then the original method on the superclass.

Figures 4 and 5 exemplify the application of some of these changes.

```
public class B {
    public B() {
        System.out.println(toString());
    }

    public String toString() { ... }
}
```

Fig. 4. Transactified `B` class.

```
public class B {
    public B() { ... } // Same as original
    public String toString() { ... } // Same as original

    public B(SpeculativeCtorMarker marker) {
        SpeculationControl.nonTransactionalActionAttempted(...);
        System.out.println(toString$speculative());
    }

    public String toString$speculative() { ... }
}
```

Fig. 5. `B` class after introduction of `$speculative` versions of methods. `java.lang.System` is an unmodifiable class, so access to its field `out` is considered a nontransactional action, as is the invocation of `println` on the `java.io.PrintStream` it contains.

[3] We consider accesses to unmodifiable classes to be nontransactional, including their fields.

2.3 Doing speculation

After the transactification and the addition of support for handling nontransactional operations, a final round of modifications for speculation is introduced. These allow the speculation system to know when it can spawn a speculative execution, and when the speculation results should be applied or discarded.

Currently, JaSPEx speculates only on method executions: When a method is invoked, some of the methods it invokes may be run speculatively. Speculation is only considered for methods in which their arguments can either be determined statically or are simple dynamic cases like arithmetic operations or parent method argument accesses. Note that method invocations inside loops are speculated at most once; further invocations are executed normally in the thread of the caller (but may still spawn speculations of their own). As an example, consider a recursive implementation of the Fibonacci function shown in Figure 6.

At each call to `fib`, JaSPEx speculatively launches the execution of `fib(n-1)` and `fib(n-2)` and then proceeds with the execution of the method: In the case where `n` $\leq$ `1`, the speculative executions that may be running are discarded; otherwise, their results are retrieved and the transactions that they are running in are committed, if possible.

Speculative methods call the JaSPEx runtime when they are started, before they terminate, and to get results from speculative executions. When a method starts, it calls the method `SpeculationControl.entryPointReached`, passing as arguments an entry-point id that uniquely identifies each speculative method, and an array of arrays with the arguments for each function call that is to be executed speculatively within that method. For instance, in the `fib` example, we will execute speculatively `this.fib(n-1)` and `this.fib(n-2)`.[4] Thus, `entryPointReached` will receive an array `arr` of type `Object[2][]`, where `arr[0]` contains the arguments `this` and `n-1`, and `arr[1]` contains `this` and `n-2`. As a result of the call to `entryPointReached`, an instance of `SpeculationId` is returned, which identifies the current dynamic execution context uniquely.

Before a method exits, a call to `SpeculationControl.exitPointReached` is made, to inform the runtime that the method will terminate, and that speculative executions that might be queued or running for this method should be discarded. The current form in which the call to `exitPointReached` is injected does not yet take into account exceptions; support for this is considered future work.

[4] Because `fib` is not a static function, each recursive call also implicitly includes as an argument the current object instance, `this`.

```
public int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

Fig. 6. Fibonacci function.

```
public int fib$speculative(int n) {
    SpeculationId specId =
        SpeculationControl.entryPointReached(ENTRY_POINT_ID,
            new Object[] { new Object[] { this, n-1 },
                           new Object[] { this, n-2 } });
    if (n <= 1) {
        SpeculationControl.exitPointReached(specId);
        return n;
    }
    Future f0 = SpeculationControl.getResult(specId, INV_ID_0);
    Future f1 = SpeculationControl.getResult(specId, INV_ID_1);
    int temp = f0.get() + f1.get();
    SpeculationControl.exitPointReached(specId);
    return temp;
}
```

Fig. 7. The speculative version of the Fibonacci function. The symbols INV_ID_* identify the function calls that they replace: In this case, INV_ID_0 represents the call to `fib(n-1)`, whereas INV_ID_1 represents the call to `fib(n-2)`.

Method invocations for methods that are executed speculatively are replaced by calls to `SpeculationControl.getResult`, which, given the current `SpeculationId` and an identifier that identifies the function call, returns a `Future` object that represents the result of the speculative execution. Finally, to obtain the result, `get()` is called on the Future; if the underlying method execution resulted in an exception being thrown (an instance of `java.lang.Throwable` or any of its subclasses), that exception will be rethrown by `get()`.

Figure 7 shows the `fib$speculative` method with these modifications.

2.4 Runtime control

As seen in the previous sections, calls to methods of the class `SpeculationControl` are added at various points of the speculative methods, allowing control of speculation start and end, decision on how to proceed when nontransactional actions need to be executed, and fetching of results from speculative executions.

A speculation starts with a call to `SpeculationControl.entryPointReached` which, as seen before, receives an entry-point id and an object array containing arguments to be used for speculative calls. The entry-point id is used to access a list of instances of `java.lang.reflect.Method`, each of which corresponds to a method that is going to be executed speculatively.[5] JaSPEx generates a new task for each element in this list: Each task will compute a call to the corresponding method with the appropriate arguments. For instance, for the execution of `fib(n)`, two smaller tasks are generated, representing the calls to `fib(n-1)` and `fib(n-2)`. These tasks are queued for execution by worker threads.

[5] This list is only generated the first time that a speculative method is executed.

When a worker thread picks up a task, it starts a new STM transaction and uses reflection to invoke the method with the supplied arguments. Because the method executes within an STM transaction, none of its changes are visible to the outside until the transaction commits. Moreover, if, during the method execution, it tries to execute a nontransactional action, it stops and waits for permission to commit its current STM transaction – it waits until it can run on normal, sequential program order, so that it cannot be aborted. If, instead, the method terminates with a return value or an exception, it also waits for permission to be committed. Finally, as a method running speculatively may also cause other speculative execution tasks to be created, when a method wants to give permission to commit to a method speculation that it started, it also has to wait for permission to commit its own transaction first.

Permission for a speculative task to commit is given only when the method `get` is called on the `Future` representing the task (itself a result from a call to the method `SpeculationControl.getResult`) and that call is made by the thread currently running in the normal program order. This scheme results in speculative transactions being committed in original, sequential program order, as expected. If a conflict is detected when trying to commit a transaction, the task is aborted and reexecuted; this time, it will commit for sure, because it is executing in the original program order.

3 Experimental results

We now present some preliminary results of the automatic parallelization performed by JaSPEx. These results were obtained on a dual-quadcore system with two Intel Nehalem-based Xeon E5520 processors, running Ubuntu Linux 9.04, and Java SE version 1.6.0_13.

As `fib` does very little computation at each step, we have modified it to do speculative execution only up to a threshold, and from then on to run the rest of the computation entirely without speculative execution on the same thread. Figure 3 presents the time needed for calculating `fib(50)` using this version with 1 to 8 cores.

These are very preliminary results, but they are encouraging, showing that it is possible to automatically extract parallelism from a sequential program with the approach that we propose. Still, we believe that further optimizations, specially to the way threads are spawned and terminated, will provide better results.

4 Related work

Transactional Memory was initially proposed by Herlihy and Moss [12] as a multiprocessor architecture capable of making lock-free synchronization as efficient and easy to use as conventional techniques based on mutual exclusion. The implementation was based on extensions to multiprocessor cache-coherence

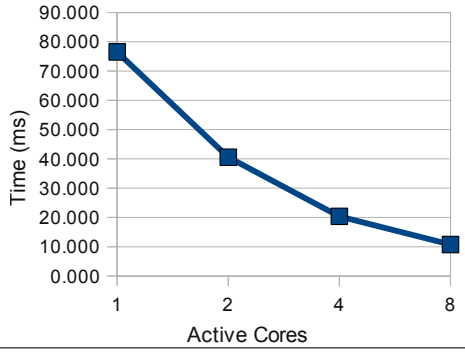


Fig. 8. Time for calculating `fib(50)` using speculative parallelization, as we increase the number of available cores.

protocols, addition of some new instructions to the processor, and a small transactional cache where transactional changes were kept prior to committing.

Software transactional memory was later introduced as an alternative to hardware transactional memory [6] that could be implemented using Load-Linked/Store-Conditional of a single memory word, as provided by most current hardware architectures. The Dynamic Software Transactional Memory (DSTM) [7] was the first unbounded STM, allowing it to be used in the implementation of dynamically-sized data structures such as lists and trees.

Many hardware-supported thread-level speculation (TLS) systems have been proposed by researchers. POSH [3] presents a TLS infrastructure on top of the GNU Compiler Collection (GCC), composed of a compiler and a profiler; it parallelizes applications by analyzing the source code and using heuristics to identify tasks, which can be further refined by using the profiler. In [4], the authors present a reverse compilation framework that translates binary code to static single assignment (SSA) form, from there performing optimizations and adding support for speculative execution. Jrpm [5], the Java runtime parallelizing machine is a Java virtual machine that does TLS on a multiprocessor with hardware support. It analyses buffer requirements and inter-thread dependencies at runtime to identify loops to parallelize. Once sufficient data is collected, the selected loops are dynamically recompiled. As Jrpm works at the Java bytecode level, no changes need to be made to the source binaries or code.

The primary difference between these systems and JaSPEx is our use of a software-based TM. Because a software-based TM has no inherent limits to transaction duration and size, we expect to be able to extract more parallelism from an application, parallelism that is available only at a higher level of the application.

The Java Fork/Join Framework [13] is a framework due for inclusion on the upcoming Java 7 that supports a style of parallel programming where problems are solved by recursively splitting them into subtasks, which can then be exe-

cuted in parallel. JCilk [14] is a Java-based language for parallel programming that supports a similar fork/join idiom, but includes very strict semantics for exception handling, aborting of side computations, and other interactions between threads that try to minimize the complexity of reasoning about them. Welc et al. [15] introduce safe futures for Java, which are futures that work as semantically transparent annotations on methods, where execution of a method can be replaced for execution of a future, but where sequential execution semantics are respected, and observed behavior of serial and concurrent tasks are the same; the implementation includes features very similar to those provided by STMs.

Our current implementation is very similar to the fork/join style of programming: Speculative tasks are created at the beginning of `$speculative` methods, and the joins are done at the original method call sites. Unlike other fork/join-style frameworks [13,14], though, where algorithms need to be explicitly modified to use fork/join calls, our framework tries to do a similar conversion automatically, including detection of conflicts between multiple tasks, which these frameworks also leave up to the programmer. The work of Welc et al. [15] is also similar to ours, because it allows multiple parts of the code to run speculatively in parallel, and includes STM-like support for aborting speculative executions if conflicts are detected. Unlike ours, however, the program needs to be manually modified to use the safe futures, and depends on their modified JVM for execution.

5 Conclusions and future work

In this paper we proposed to use an STM-based approach to thread-level speculation, so that we may extract more parallelism from sequential programs, benefit from the results of the transactional memory research community, and target current hardware.

We have incorporated our proposal into a running system – JaSPEx – that automatically parallelizes a program that was compiled to run in the Java Virtual Machine. To accomplish that, JaSPEx transforms the program, without the intervention of the programmer, so that some parts of it may execute speculatively. One of the challenges in this transformation is the transactification of the program. In this work we describe some of the difficulties inherent to the transactification of a JVM program if we have no support from the JVM runtime. Because of those difficulties, the transactification performed by JaSPEx is currently limited, but we intend to address that problem in the future by supporting the transactification at the JVM-runtime level.

In its current state, JaSPEx shows promising results – obtaining linear speedup on a recursive implementation of the fibonacci function – even though it has not been tested on realistic benchmarks, yet. Nevertheless, in the future we hope to obtain further speedups by reducing overheads in task creation, commit, and abort; by implementing a more dynamic system that gathers statistics on speculation duration and success rates, with the objective of avoiding method speculations for very small methods and for methods with high abort rates; and by

further optimization of the JVSTM for our use-case, thereby reducing overheads in transactional execution of applications.

References

1. Blume, B., Eigenmann, R., Faigin, K., Grout, J., Hoeflinger, J., Padua, D., Petersen, P., Pottenger, B., Rauchwerger, L., Tu, P., et al.: Polaris: The Next Generation in Parallelizing Compilers. In: Proceedings of the Seventh Workshop on Languages and Compilers for Parallel Computing. (1994)
2. Wilson, R.P., French, R.S., Wilson, C.S., Amarasinghe, S.P., Anderson, J.M., Tjiang, S.W.K., Liao, S.W., Tseng, C.W., Hall, M.W., Lam, M.S., Hennessy, J.L.: Suif: an infrastructure for research on parallelizing and optimizing compilers. SIGPLAN Not. **29**(12) (1994) 31–37
3. Liu, W., Tuck, J., Ceze, L., Ahn, W., Strauss, K., Renau, J., Torrellas, J.: POSH: a TLS compiler that exploits program structure. In: PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming, New York, NY, USA, ACM (2006) 158–167
4. Yang, X., Zheng, Q., Chen, G., Yao, Z.: Reverse compilation for speculative parallel threading. Parallel and Distributed Computing Applications and Technologies, International Conference on **0** (2006) 138–143
5. Chen, M., Olukotun, K.: The Jrpm system for dynamically parallelizing Java programs. In: Proceedings of the 30th annual international symposium on Computer architecture, ACM New York, NY, USA (2003) 434–446
6. Shavit, N., Touitou, D.: Software transactional memory. Distributed Computing **10**(2) (1997) 99–116
7. Herlihy, M., Luchangco, V., Moir, M., Scherer III, W.: Software transactional memory for dynamic-sized data structures. In: Proceedings of the twenty-second annual symposium on Principles of distributed computing, ACM Press New York, NY, USA (2003) 92–101
8. Bruneton, E., Lenglet, R., Coupaye, T.: ASM: a code manipulation tool to implement adaptable systems. Adaptable and extensible component systems (2002)
9. Cachopo, J., Rito-Silva, A.: Versioned boxes as the basis for memory transactions. Science of Computer Programming **63**(2) (2006) 172–185
10. Cachopo, J.: Development of Rich Domain Models with Atomic Actions. PhD thesis, Technical University of Lisbon (September 2007)
11. Bracha, G.: Generics in the Java programming language. Sun Microsystems, java. sun. com (2004)
12. Herlihy, M., Moss, J.: Transactional memory: architectural support for lock-free data structures. In: Proceedings of the 20th annual international symposium on Computer architecture, ACM New York, NY, USA (1993) 289–300
13. Lea, D.: A Java fork/join framework. In: Proceedings of the ACM 2000 conference on Java Grande, ACM New York, NY, USA (2000) 36–43
14. Danaher, J., Lee, I., Leiserson, C.: The JCilk language for multithreaded computing. In: OOPSLA 2005 Workshop on Synchronization and Concurrency in Object-Oriented Languages (SCOOL). (2005)
15. Welc, A., Jagannathan, S., Hosking, A.: Safe futures for Java. In: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications, ACM New York, NY, USA (2005) 439–453

Domain-Specific Languages: A Theoretical Survey

Nuno Oliveira[1], Maria João Varanda Pereira[2], Pedro Rangel Henriques[1], and Daniela da Cruz[1]

[1] University of Minho - Department of Computer Science,
Campus de Gualtar, 4715-057, Braga, Portugal
{nunooliveira, prh, danieladacruz}@di.uminho.pt
[2] Polytechnic Institute of Bragança
Campus de Sta. Apolónia, Apartado 134 - 5301-857, Bragança, Portugal
mjoao@ipb.pt

Abstract. Domain-Specific Languages (DSLs) are characterized by a set of attributes that make them different and easy to use when compared to General-purpose Programming Languages (GPLs). The fact of being tailored for a specific domain rises many advantages on their usage, however special care must be put in their conception and implementation.
The purpose of this paper is to provide a survey on DSLs, enhancing their characteristics that make clear the advantages and disadvantages of their usage and make challenging their implementation. We also focus on the development methodologies that have been used to create the thousands of DSLs that exist today, which are a powerful alternative to GPLs.

1 Introduction

No matter what is done, computers will always understand things in terms of 0's (zeros) and 1's (ones). These *things* they understand are human-made programs. Humans, although capable of doing it, are not proficient in *speaking* that binary language. However, the computer can be taught to translate any language into its preferred idiom. For this reason, humans do not need to go down to binary level. Instead, they can keep the way they talk to computers at a very perceptible level by rising the abstraction level of the language they use.

Programming is a computer-oriented task, but other actors are also involved on it. One of the most important concerns when developing a software piece is about its future, namely, its maintenance. It is not a computer that will maintain the software, but a human; so humans must be taken into consideration when choosing the programming language and writing the code. Thus, the terms and concepts used in the programming language (its syntax and semantics) should be close to those persons that have to maintain the software.

Two types of languages are mainly used to write programs for computers: GPLs and DSLs.

There is not a precise definition for GPL [1]. GPLs are tailored to be used to solve any kind of problem, no matter the area or domain this problem fits into. Normally they are the programmer's preference for the communication with the computer. Many relevant factors contribute to this choice. As they are general purpose and widely used,

these languages are adopted by the majority of programmers. The existence of a large community of experts in one GPL also plays an important role in such adoption. But issues like the maturity of a language the availability of well tested optimizing compilers, and the existence of good development tools or environments [2], are the more crucial for the referred preference. But on the other hand, GPLs also have many drawbacks when regarding other aspects like writing and reading (the learning curve has a long setup time), or understanding their programs. Concerning the former aspect (writing and reading), they always imply vast programming expertise. Hence, not every one is able to use them properly. Concerning the latter aspect (program comprehension), GPLs are hard to understand because on the one hand, their syntax and semantics is not obvious due to their generality, and on the other hand they commonly address implementation particularities that are closer to the machine's way of working than to the human's way of thinking. Even following different programming paradigms or exhibiting various syntactic constructions, which aims at raising the abstraction level, is not enough to overcome the difficulties observed on comprehending GPL programs.

DSLs [3] can be defined by the following sentence:

> *A domain-specific language is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.* [4]

As these languages are conceived for a specific domain, it is easy to absorb the main concepts and features inherent to this domain, and bring them to the language constructors [5]. This should result in a syntax and semantics capable of effectively raising the abstraction level of the programming [6] task; that is, the notation addresses a higher level of abstractness, increasing the distance to the machine's way of working and shortening it to the human's way of thinking. Some examples of DSLs include languages like LaTeX for text processing, DOT for graphs drawing, SQL for databases and so on.

The reminder of this paper is structured in five sections. First, in Section 2, the characteristics of these languages are addressed. The main differences to GPLs are discussed in Section 3. The characteristics of these languages are intrinsically related to the advantages and disadvantages that they bring to their usage and development. These pros and cons are discussed in Section 4. Approaches and methodologies to the development of DSLs are discussed in Section 5. Finally, in Section 6, the paper is concluded.

2 DSLs Characteristics

GPLs are perfectly established in the software development life-cycle. Their characteristics are so widely spread among the software engineers community that are regarded as a natural thing, therefore, literature badly addresses this matter. However, with DSLs does not happens the same. There is the necessity of enhancing their characteristics in order to defend their usage instead of GPLs. In part, this happens because of the research and studies made on DSLs during the last ten years [7, 8, 5]; what reveals the

importance that these languages are achieving in software engineering. Because of that, the characteristics of DSLs are worthwhile to know and understand. The following paragraphs present those characteristics and justifies them.

Each DSL has unique characteristics, however the major part is common. In this context, it is possible to create a shelf for these languages, and evaluate and explain their common characteristics. In [9], the authors used the Cognitive Dimensions of Notations Framework (CDF) [10, 11] to accomplish a speculation-based evaluation of DSLs. In this section, the intention is to do a similar analysis. However in this case, the aim is to enhance, describe and point reasons for the characteristics of DSLs, regardless of the CDF.

Normally, DSLs are *small*. As these languages are tailored to deal with the problems of a specific domain, their designers can grab only the essential features and concepts of that domain, and manage them to create a small and restricted notation. That small notation allows the specification of solutions to solve the problems, instead of programming them, unlike what happens when dealing with the major part of GPLs. So, DSLs are more *declarative* or *descriptive* than imperative languages [4].

Moreover, they are *abstract* [12] and *expressive* [8]. When analyzing and designing a DSL, engineers should be aware of the domain where the language will be applied. The semantics of the domain should be implicit in the language notation [13]. The latter means that abstraction should be brought to the notation of the language; that is, the low level notions of how something is done should be encapsulated by a high level notation, expressing, for each sentence or statement, the precise purposes of their existence and usage. Indubitably, this allows their users to easily create mappings between the syntax of the language and the objects of the problem domain.

The concentration on the definition of a notation that would only express concepts of a single application domain, brings the possibility of sharpening edges on the language, and make it more and more *efficient* on various directions. One of these directions is the efficiency on being read and learned by the domain experts [6]. Domain experts are persons with great knowledge on a given domain, but, normally with any or little expertise on programming. Thus, given the abstractness and the expressiveness of DSLs, they can easily read programs, and learn the languages in order to specify the programs with efficiency, or in another words, with little time spent. Another facet of that efficiency can be observed on the tools that give support to the language. Their processors, for instance, can be improved to offer better results, as the domain is restricted and the knowledge is centralized.

3 Dichotomies on DSLs

DSLs are dichotomous. Their definition and characteristics divide the languages into two parts that can be regarded as a good and a less-good part. When analyzing these dichotomies it is inevitable the comparison with GPLs. For this discussion, the following pairs of characteristics are taken into account: i) *Abstractness vs Concreteness*; ii) *Low-Level vs High-Level* and iii) *Expressiveness vs Computational Power*.

3.1 Abstractness *versus* Concreteness

Abstractness and concreteness are subjective terms, because depend on the point of view, and on the objects that are being related. However, as stated before, programming is a human-oriented task in these days; so the perspective is always on the human side, and not on the computer's. Nonetheless, the computer is the object to relate with, in order to perform an analysis. So, what should be answered when is asked whether DSLs are abstract or concrete? A language is more abstract the more it hides the operational semantics [14] from the final user. One of the main efforts during the construction of a DSL, is to encapsulate the semantical knowledge in the language notation, in order to have it implicitly, instead of explicitly, as happens with GPLs. For this reason DSLs are more abstract than GPLs, which, by exclusion of parts, are more concrete.

In fact, as *Deursen* and *Klint* observed [15], the knowledge about the domain is concentrated in the language notation, and the knowledge about the implementation (the operational semantics of the language) is delegated to the compilers, processors or other related tools that give support to the language usage.

There are gains and losses on this matter. But surely the gains overcome the losses. Section 4, embodies a discussion about the advantages and disadvantages of using DSLs; there, the gains and losses of the abstract characteristic of DSLs, are addressed.

3.2 Low-Level *versus* High-Level

These dichotomous words are very similar to concreteness and abstractness, respectively. As a matter of fact, they depend on them. A low level language is a language that takes its user into a thinking level that is known to be unusual for a human beings. That is, the handling of constructors and abstractions of a low level language imply the presence of specialized knowledge beyond the empirical knowledge on an application domain. On the other hand, high level languages, remove from their notations explicit implementation aspects. Users are able, then, to handle the language constructors at a more rational level.

DSLs are high level languages, when comparing with GPLs. The abstractness of DSLs enable the rise of this level, as the semantics of the language are implicit in the constructors and abstractions that the notation of the language presents. This way, the user's concerns are focused on issues associated with the problem and not with the solution. The gains on following this philosophy are great; using high level languages (DSLs) that allow the focus on the problem and not in the solution, can be profitable at earlier stages of the software life-cycle [16], like the requirements analysis and management [17].

3.3 Expressiveness *versus* Computational Power

A language is said to be expressive when its notation helps the user on creating mappings between the program and problem domain concepts, without resorting to documentation of the software pieces, whether it is internal (comments, annotations and so forth) or external (user manuals, implementation reports and so forth). Such characteristic is observable when the language is easy to learn, and their programs are easy to write, read and understand.

On the other hand, the computational power of a language is related with the possibility of specifying multiple and different computations relying on the same vocabulary and structures, offered by the language. Also it is manifested when the notation allows the definition of similar computations, using different approaches.

DSLs are usually, more expressive than GPLs; on the other hand, GPLs offer a greater computational power. The major difference between these types of languages is precisely this dichotomy. However, it was not always true. Languages like Fortran or Cobol, can be regarded, nowadays, as GPLs, because of their computational power, but they were first constructed to fulfill requirements on the mathematics and the business areas, respectively [4]. That is, they were tailored for a single application domain.

The expressive power of DSLs make them very objective languages, in the extent that the user would know what happens when a statement of a program is interpreted or executed. The vocabulary of these languages store knowledge of the application domain, and mask behavioral aspects of this same domain. As they are used to cope with the aspects of a single domain, they do not need extra computational power to extrapolate for other domains. The language constructors must encapsulate precisely the behavior required for the concepts of the domain, with which they are associated.

As final words about this dichotomy, *Ladd* and *Ramming* [18] state that DSLs should not be designed to describe computations, but to express useful facts from which one or more computations can be derived. This express precisely what was said about DSLs through out the present section.

4 Advantages and Disadvantages

All the characteristics of DSLs, listed and explained before, imply a group of advantages, and obviously, some of them also a set of disadvantages.

In this section, an impartial and literature-based overview on this matter is given. In order to proceed, it is important to settle down two perspectives on this discussion. On the one hand, there is the *language usage* perspective, which is related with the usability of DSLs to produce, maintain, evolve or comprehend programs or specifications. On the other hand, there is the *language development* perspective, which concerns the implementation of processors (compilers or interpreters, editors, debuggers, animators, etc.) for the new DSLs.

4.1 On DSLs Usage Perspective

The most claimed advantage of using DSLs is the possibility of integrating domain experts in later stages of the software development life-cycle [15, 19]. Normally, domain-experts are very required in the analysis phase, and their functions end right there. But a few times, a new overview on the conceptual aspects concerning the software in production, is needed. Since the usage of GPLs require good programming skills, the domain-experts, who are not proficient on that area, little work can do on this matter. However, using DSLs they can steer the flow on the programming tasks and they can even give a hand on specifying the programs.

In this context, DSLs are also appropriated to diminish the distances that exist between the conception and implementation phases of the software development process. Also, this leads to the creation of new software development methodologies, meeting the requirements that the domain imposes [20].

Concerning the *software development process*, it is possible to sub-divide the present perspective into two parts: (i) the initial phases on the development process and (ii) the maintenance phase.

Concerning the former sub-perspective, (i), DSLs ease and increase the speed on the construction of software pieces [15]. *Kieburtz et al.* [21] also defined a set of advantages, that, in some extent, empower the existence of the last addressed advantage: they claim that DSLs enhance the flexibility, productivity and usability. The justification for these advantages is trivial by regarding the characteristics of DSLs presented before. All they are due, mainly, to the expressiveness and abstraction of these languages, which are, indeed, the main characteristics. Also, the usability of these languages can be justified by the fact of being small and easy to read and write.

DSLs also make easier the phase of testing in the software development process. In this context, non-traditional methods like [22] can be followed in order to test these programs.

Regarding the latter sub-perspective, (ii), there are many advantages. The main one is that using DSLs the software maintenance is simplified [15]. These languages enhance the comprehensibility of programs and specifications [7, 5]; which is not any novelty, because the easiness of maintaining a piece of software implies the easiness of comprehending the program specifications. Another important aspect of these languages is that, in many cases, they provide self-documentation, what avoids the search for documentation resources that may be unavailable. Together, these three aspects diminish the costs of engineering and reengineering, and increase reliability and repairability on the software constructed with DSLs [23].

DSLs are claimed to be a good approach for software reuse [24] — another advantage from the usage of DSLs. In this context, not only the pieces of software are reused, but also the knowledge embodied in the language.

Until now, only advantages were pointed. However DSLs have also some bad things associated, concerning their usage. Those presented below derive from the fact that DSLs are multi-characteristic languages. It is known that, several and distinct characteristics can difficult the adherence of users, because not all of them are used to cope with such variety of languages, and some programmers claim that do not need to learn many DSLs when already know one GPL. The point is, if there is not adherence on a DSL, it would not have any success, and its evolution or work done upon them, like tool support construction, would be tasks with no future; on the other hand, the adherence implies teaching costs, which can be very high [15]. Although the latter has been pointed as a disadvantage, it is believed that is easy to learn a DSL without effort.

Moreover, this variety reduces the easiness of interoperability with other well established languages [19]. This is obviously a disadvantage, because in real projects, several domains can be addressed and, thus, several DSLs can be used; and even the combination of DSLs with GPLs is a reality. So, without interoperability between them, DSLs can fall into disuse.

4.2 On DSLs Development Perspective

The development of DSLs allow optimization and validation at domain level [25]. The optimization of a language concerns with details of implementation at machine level, e.g. managing the memory allocation. As was stated before, these kind of implementations are done at language development time, and not at programming time, providing abstractions that encapsulate these details. So, at programming time, the user is able to build specifications and optimize them at domain level, because the optimization at machine level, is already done. For instance, in C, memory allocation is an indispensable task, but it takes the user to cope with details that are, most of the times, beyond their capabilities.

But this raises the major part of the disadvantages of developing DSLs. Firstly, in order to develop a language, the engineer must be expert on both the domain of application and compilers engineering [15]. Most of the times, the language engineers are not proficient on the problem domain, so creating a language is not a single-man task; domain experts are needed to steer the DSL requirements [26], and language engineers are needed to concretize the requirements into an abstraction capable of coping with the domain concepts.

This fact empowers the consumption of time and money on the several phases of the language development (design, implementation and maintenance) [4]. In fact, developing a language requires knowledge about programming in GPLs, on most part of the cases. The issues related with the maintenance of GPLPs are widely known. So, maintaing DSLs and their associated tools, like compilers, processors or interpreters, can be a very complicated and difficult task. Not for so little times, this leads to low number of tool supporting; and when there are tools, they may not follow up the evolutionary trends of the language [19].

Missing tool supporting for monitoring, debugging and other necessary tasks for DSL users is, indubitably, a great disadvantage. Without tools, the availability of DSL is limited [4], what leads to low number of adherents on such language; and the disadvantages of the last argument were already addressed in the previous section.

The disadvantages of the present perspective, can be attenuated regarding the methodologies used for the development of DSLs. Section 5 gives a little survey on some development methodologies that were successfully used to implement DSLs, over the times.

5 Developing DSLs

Language engineering is an old discipline, perfectly established as a branch of the software engineering. The development of DSLs is just a small part on that branch. Unlike the development of GPLs, which is, normally, based on compiler techniques [27], the development of DSLs follows several methods and techniques, including also the techniques used for creating GPLs [8].

Each development methodology is well supported by tools. Not that all the tools were tailored to cope with the development of DSLs but they are successfully and easily

adapted to cope with it. The DSLs that outcome from these different methodologies, are given a classification name. But regardless of that classification, they keep the same characteristics listed before and follow the same CDs.

In general, five main steps are pointed to be essential on DSL development: decision, analysis, design, implementation and deployment [8, 4]. The decision phase concerns with the analysis of pros and cons about developing or not a new language. The analysis is the phase where domain experts gather knowledge related to the domain, relying on domain analysis methodologies [28–30]. The design step is concerned with the choice and adoption of patterns and conceptual implementation decisions. The implementation phase, is related to the concrete construction of the DSL, using the patterns and implementation approaches adopted in the design phase. The deployment (or distribution) step is when the language is ready to be used, and is made available for general usage.

During these phases, patterns can be used to ease the process of language development. *Spinellis* [31], reinforced later by *Mernik et al.* [8], defined a set of several design patterns. These patterns are basis for implementation approaches. In [7], three main shelves were identified to contain such approaches. In the sections below are presented the two main containers and the approaches contained in each one.

5.1 Complete Language Design Approaches

The approaches contained in this shelf involve the creation of a language from the zero. This requires the definition of the language syntax (commonly achieved by using Backus-Naur Form (BNF) or Extended Backus-Naur Form (EBNF) notations), the specification of the semantics, and the translation into target code. *Compilation* and *Interpretation* are the main approaches within this container.

Compilation — It is the most traditional approach to implement a language, no matter the language is a DSL or a GPL. The language constructs are analyzed and synthesized into a target code, that can be machine code or a common GPL language code, that is to be executed by a machine.

Interpretation — The difference to the compilation approach is that the language constructs are recognized but not executed at machine level; instead they are interpreted.

These implementation approaches are supported by compiler-compilers (tools for compiler generation) that are widely available for many years. The Lex and YACC [32, 33] system, used for the development of any kind of language, is one of the most popular. More recent and similar tools (in the extent that are not only focused on the development of DSLs) are available: JavaCC[3], SableCC [34], based on translation grammars, and LISA [35], AntLR [36], JastAdd [37], based on Attribute Grammars (AG) [38], are some examples. On the other hand, there are tools specialized on the construction of compilers and interpreters for DSLs. Draco [39], ASF+SDF [40], Kephera [41], Kodiyak [23], InfoWiz [42], are some examples of such tools.

The languages created with these approaches are said to be *external languages* [43], because they are independent languages; that is, the DSL is completely designed and implemented from the scratch, not relying on any pre-existent language.

[3] Home page: https://javacc.dev.java.net/

5.2 Languages Extension Approaches

In this container are placed the approaches for DSLsimplementation using GPLs, or components of these languages, as a start point. In this context, two approaches are considered to be the most used:*Embedding* and *Extensible compiler/interpreter* approaches.

Embedding — In these approaches, the developer does not need to have compilers expertise, because all the work of semantics verification and target code transformation is delegated to the compiler of the base language. Nevertheless, the constructs of the new DSL must be designed, using the base language syntax. The construction of an Application Programming Interface (API) for a given application domain, is considered a concretization of this approach, and is the several times used.

The advantages of embedding languages are considerable. From the fact that it needs no compiler knowledge, to the fact that the compiler of the base-langauge is completely reused; from the point that the development of the language needs only knowledge on the domain and on the base language, to the point that, for usage purpose, it is not need to know the underlying language.

The languages implemented with this approach are called *internal* or *embedded languages* [12]. Almost any GPL can be used as base-language, but some have more appropriated characteristics and are frequently used: Ruby [44], Python [45], Haskell [12, 46], Java [47], C++ [48] and Boo [43] are some examples of utilization.

A specialization of the embedded languages was introduced by *Fowler* and *Evans* [49], and is called *Fluent Interfaces*. These languages still being classified as embedded, but their construction upon a GPL follows a methodology enabling a more flexibility in the syntax. So writing operations is no more than chaining function calls, what allows a more fluent specification.

Extensible compiler/interpreter — This approach is very similar to the compilation approach presented in Section 5.1. The main difference is that an existent compiler of a GPL is extended with domain-specific aspects in order to add domain-specific constructs to the underlying language, rather than creating the compiler from the scratch. This way, the task of creating the language is easier than using the other approach.

Summing up this discussion about the implementation of DSLs, *Kosar et al.* [5], concluded that the extension of languages by the embedding approach is the most efficient and effective approach to be used when developing DSLs. They also claim that a great alternative to this approach is to use the compilation approach, because of the possibility of handling minor aspects and having more control over the language implementation.

Another shelf identified is using Commercial Off-The-Shelf (COTS) products. The idea on this approach is to specify the language structural aspects in terms of these tools [7]. As this approach is not so consensual, no more details about it are given.

6 Conclusion

The usage of Domain-specific Languages (DSLs) is not a novelty. Since the beginning of programming languages, DSLs have been created to focus the programmer on the particularities of a concrete problem domain, and not on the programming details. Thus,

a great number of DSLs have been tailored for a considerable amount of domains. *Mernik et al.* [8] provide several examples of existing DSLs and identify, also, their application domains.

The fact of DSLs being designed for a specific domain confers them a reasonable number of characteristics that make them different from GPLs and more competitive. However, to get efficient implementations, the development of their processor (compiler, interpreters, etc.) is not an easy task and requires systematic approaches based on traditional compiler construction technology. Nevertheless, these difficulties can be soften regarding the variety of implementation approaches that have been successfully used.

Although not yet corroborated by the literature, the community believes that DSLs are much more user-friendly, and their programs are easier to comprehend, when comparing with GPLs. Experiments and work on program comprehension for DSLs and their usability have been taken off in the context of DSLpc[4]. There are already some raw results on this matter, described in some papers recently submitted to conferences on the area.

The objective of this paper was to summarize a review of DSLs literature. Particular attention was paid to the theoretical perspective associated with these languages, rather than to the practical one, as have been done for the last ten years. Basilar characteristics, advantages and disadvantages were pointed out. In addition, approaches and methodologies for their development were surveyed, for a complete overview of the topic under discussion.

References

1. Watt, D.A.: Programming language concepts and paradigms. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1990)
2. Hutchings, B.L., Nelson, B.E.: Using general-purpose programming languages for FPGA design. In: DAC '00: Proceedings of the 37th conference on Design automation, New York, NY, USA, ACM (2000) 561–566
3. Visser, E.: WebDSL: A case study in domain-specific language engineering. In Lammel, R., Saraiva, J., Visser, J., eds.: Generative and Transformational Techniques in Software Engineering (GTTSE 2007). Lecture Notes in Computer Science, Springer (2008)
4. van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. ACM SIGPLAN Notices **35** (2000) 26–36
5. Kosar, T., López, P.E.M., Barrientos, P.A., Mernik, M.: A preliminary study on various implementation approaches of domain-specific language. Information and Software Technology. **50**(5) (April 2008) 390–405
6. Consel, C., Latry, F., Réveillère, L., Cointe, P.: A generative programming approach to developing dsl compilers. In Gluck, R., Lowry, M., eds.: Fourth International Conference on Generative Programming and Component Engineering (GPCE). Volume 3676 of Lecture Notes in Computer Science., Tallinn, Estonia, Springer-Verlag (sep 2005) 29–46
7. Wile, D.S.: Supporting the dsl spectrum. Journal of Computing and Information Technology **9**(4) (2001) 263–287

[4] DSLpc is a collaborative project between Portugal and Slovenia, and stands for *Program Comprehension for Domain-specific Languages*. More details can be found in http://epl.di.uminho.pt/~gepl/DSL/

8. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Computing Surveys. **37**(4) (December 2005) 316–344
9. Pereira, M.J.V., Mernik, M., da Cruz, D., Henriques, P.R.: Program comprehension for domain-specific languages. ComSIS – Computer Science an Information Systems Journal, Special Issue on Compilers, Related Technologies and Applications **5**(2) (Dec 2008) 1–17
10. Blackwell, A., Britton, C., Cox, A., Green, T.R.G., Gurr, C., Kadoda, G., Kutar, M., Loomes, M., Nehaniv, C., Petre, M., Roast, C., Roe, C., Wong, A., Young, R.: Cognitive dimensions of notations: Design tools for cognitive technology. In: Cognitive Technology: Instruments of Mind. Springer-Verlag (2001) 325–341
11. Green, T.R.G., Blandford, A.E., Church, L., Roast, C.R., Clarke, S.: Cognitive dimensions: achievements, new directions, and open questions. Journal of Visual Languages & Computing **17**(4) (August 2006) 328–365
12. Hudak, P.: Building domain-specific embedded languages. ACM Computing Surveys **28**(4) (June 1996) 196–202
13. Hudak, P.: Modular domain specific languages and tools. In: ICSR '98: Proceedings of the 5th International Conference on Software Reuse, Washington, DC, USA, IEEE Computer Society (1998)
14. Nielson, H.R., Nielson, F.: Semantics with applications: a formal introduction. John Wiley & Sons, Inc., New York, NY, USA (1992)
15. van Deursen, A., Klint, P.: Little languages: little maintenance? Technical report, University of Amsterdam, Amsterdam, The Netherlands (1997)
16. Jackson, M.: Problem frames and software engineering. Information and Software Technology **47**(14) (November 2005) 903–912
17. Aurum, A., Wohlin, C.: Engineering and Managing Software Requirements. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
18. Ladd, D.A., Ramming, C.J.: Two application languages in software production. In: VHLLS'94: USENIX 1994 Very High Level Languages Symposium Proceedings, Berkeley, CA, USA, USENIX Association (1994) 10
19. Gray, J., Fisher, K., Consel, C., Karsai, G., Mernik, M., Tolvanen, J.P.: Dsls: the good, the bad, and the ugly. In: OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, New York, NY, USA, ACM (2008) 791–794
20. Anlauff, M., Chaussee, R., Kutter, P.W., Pierantonio, A.: Domain specific languages in software engineering (1998)
21. Kieburtz, R.B., Mckinney, L., Bell, J.M., Hook, J., Kotov, A., Lewis, J., Oliva, D.P., Sheard, T., Smith, I., Walton, L.: A software engineering experiment in software component generation. In: ICSE '96: Proceedings of the 18th international conference on Software engineering, Washington, DC, USA, IEEE Computer Society (1996) 542–552
22. Sirer, E.G., Bershad, B.N.: Using production grammars in software testing. In: PLAN '99: Proceedings of the 2nd conference on Domain-specific languages, New York, NY, USA, ACM (1999) 1–13
23. Herndon, R.M., Berzins, V.A.: The realizable benefits of a language prototyping language. IEEE Transactions on Software Engineering **14**(6) (1988) 803–809
24. Krueger, C.W.: Software reuse. ACM Comput. Surv. **24**(2) (June 1992) 131–183
25. Menon, V., Pingali, K.: A case for source-level transformations in matlab. In: PLAN '99: Proceedings of the 2nd conference on Domain-specific languages, New York, NY, USA, ACM (1999) 53–65
26. Kolovos, D.S., Paige, R.F., Kelly, T., Polack, F.A.C.: Requirements for domain-specific languages. In: Proc. 1st ECOOP Workshop on Domain-Specific Program Development (DSPD 2006), Nantes, France (July 2006)

27. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition). Addison Wesley (August 2006)
28. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute (November 1990)
29. Simos, M.A.: Organization domain modeling (ODM): formalizing the core domain modeling life cycle. SIGSOFT Softw. Eng. Notes **20**(SI) (1995) 196–205
30. Frakes, W., Diaz, R.P., Fox, C.: DARE: Domain analysis and reuse environment. Ann. Softw. Eng. **5** (1998) 125–141
31. Spinellis, D.: Notable design patterns for domain-specific languages. Journal of Systems and Software **56**(1) (February 2001) 91–99
32. Lesk, M.E., Schmidt, E.: Lex - a lexical analyzer generator. UNIX: research system **2** (1990) 375–387
33. Johnson, S.C.: Yacc: Yet another compiler compiler. In: UNIX Programmer's Manual. Volume 2. Holt, Rinehart, and Winston, New York, NY, USA (1979) 353–387
34. Gagnon, E.M., Hendren, L.J.: SableCC, an object-oriented compiler framework. In: Technology of Object-Oriented Languages, 1998. TOOLS 26. Proceedings. (1998) 140–154
35. Mernik, M., Lenič, M., Avdičaušević, E., Žumer, V.: Lisa: An interactive environment for programming language development. In: Compiler Construction. Springer Berlin / Heidelberg (2002) 1–4
36. Parr, T., Quong, R.W.: AntLR: A predicated-LL(K) parser generator. Software Practice and Experience **25**(7) (July 1995) 789–810
37. Ekman, T., Hedin, G.: The jastadd extensible java compiler. SIGPLAN Not. **42**(10) (2007) 1–18
38. Knuth, D.E.: Semantics of context-free languages. Theory of Computing Systems **2**(2) (June 1968) 127–145
39. Neighbors, J.: The draco approach to constructing software from reusable components. Readings in artificial intelligence and software engineering (1986) 525–535
40. van Deursen, A., Heering, J., Klint, P.: Language Prototyping: An Algebraic Specification Approach: Vol. V. World Scientific Publishing Co., Inc. (1996)
41. Faith, R.E., Nyland, L.S., Prins, J.F.: Khepera: a system for rapid implementation of domain specific languages. In: DSL'97: Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997, Berkeley, CA, USA, USENIX Association (1997) 19–31
42. Nakatani, L.H., Jones, M.A.: Jargons and infocentrism. In: In First ACM SIGPLAN Workshop on Domain-Specific Languages. (1997) 59–74
43. Rahien, A.: Building Domain-Specific Languages in Boo. Manning (February 2008) (Note: Unedited Draft).
44. Cunningham, C.H.: A little language for surveys: Constructing an internal DSL in Ruby. In: Proceedings of the ACM SouthEast Conference, Auburn, Alabama (March 2008)
45. Paul, R.: Designing and implementing a domain-specific language. Linux J. **2005**(135) (2005) 7+
46. Peterson, J.: A language for mathematical visualization. In: Proceedings of FPDE'02: Functional and Declarative Languages in Education. (2002)
47. Kabanov, J., Raudjärv, R.: Embedded typesafe domain specific languages for java. In: PPPJ '08: Proceedings of the 6th international symposium on Principles and practice of programming in Java, New York, NY, USA, ACM (2008) 189–197
48. Prud'homme, C.: A domain specific embedded language in c++ for automatic differentiation, projection, integration and variational formulations. Sci. Program. **14**(2) (2006) 81–110
49. Fowler, M., Evans, E.: Fluent interfaces (December 2005) Web Reference: `http://www.martinfowler.com/bliki/FluentInterface.html`; Visited on May 2009.

Sessão 1B: Sistemas Embebidos e de Tempo-Real

Flexible Operating System Integration in Partitioned Aerospace Systems

João Craveiro[1], José Rufino[1], Tobias Schoofs[2], and James Windsor[3]

[1] ⋆ LaSIGE, University of Lisbon, Portugal.
[2] Skysoft Portugal S.A., Lisbon, Portugal.
[3] ESA/ESTEC, Noordwijk, The Netherlands.

Abstract. The ARINC 653-based AIR (ARINC 653 in Space Real-Time Operating System) architecture, developed as a response to the interest of the aerospace industry in adopting the concepts of Integrated Modular Avionics (IMA), proposes a partitioned environment, observing strict temporal and spatial segregation, in which partitions are able to use different (real-time) operating systems and host applications of different criticality levels. This paper centers on recent enhancements to the AIR architecture, like the AIR POS Adaptation Layer (PAL), which aim at optimizing the development and integration processes with the flexible support to new partition operating systems (POS) in mind. We also discuss the current efforts, which already benefit from the properties of the AIR Technology, to integrate Linux as a POS, exploiting the concepts of the paravirtualization interface currently provided by the Linux kernel.

Resumo. A arquitectura AIR (ARINC 653 in Space Real-Time Operating System), baseada na especificação ARINC 653 e desenvolvida como resposta ao interesse da indústria aeroespacial em adoptar os conceitos de Integrated Modular Avionics (IMA), propõe um ambiente compartimentado e com observância estrita de segregação temporal e espacial, onde as partições podem utilizar diferentes sistemas operativos (de tempo-real) e conter aplicações com diferentes níveis de criticalidade. Este artigo centra-se em melhoramentos recentes à arquitectura AIR, como o componente AIR POS Adaptation Layer (PAL), que visam optimizar os processos de desenvolvimento e integração, com a flexibilidade no suporte a novos sistemas operativos em mente. Discutimos também os esforços actuais, que já beneficiam das propriedades da tecnologia AIR, para integrar o Linux como sistema operativo de uma das partições, explorando os conceitos da interface de paravirtualização actualmente fornecida pelo núcleo Linux.

1 Introduction

Traditional federated architectures for avionics systems are based on the distribution of avionics functions along separate collections of dedicated hardware resources. The

⋆ Faculdade de Ciências da Universidade de Lisboa, Bloco C6, Piso III, Campo Grande, 1749–016 Lisboa, Portugal. This work was partially developed within the scope of the ESA (European Space Agency) Innovation Triangular Initiative program, through ESTEC Contract 21217/07/NL/CB — Project AIR-II (ARINC 653 in Space RTOS — Industrial Initiative), URL: http://air.di.fc.ul.pt). This work was partially supported by FCT through the Multiannual Funding and the CMU-Portugal Programs.

demands of modern systems — like reducing system size, weight, and power (SWaP) — require more efficient architectures [1].

As a challenge to federated avionics architectures towards this goal, the Integrated Modular Avionics (IMA) specification defines a partitioned environment, comprising processing, communications and input/output resources, to be shared among avionics functions of different criticalities [2]. Closely related to this architecture is the Avionics Application Software Standard Interface, defined in the ARINC 653 specification [3], and the concepts of temporal and spatial partitioning.

In ARINC 653, temporal partitioning consists of the time-sliced allocation of computing resources to hosted applications, achieved through a fixed, cyclic scheduling of partitions over a major time frame (MTF). This way, strong temporal segregation is achieved, in which activities inside each partition do not affect the timeliness of activities executing inside the remaining partitions in the system. Robust spatial partitioning concerns preventing applications from accessing memory zones outside those belonging to its partition.

Having similar requirements (safety, SWaP, etc.) as avionics platforms, space missions can benefit from adopting similar approaches, which has sparked the interest of the aerospace industry, and the European Space Agency (ESA) in particular, in the concepts of IMA, and time and space partitioning [4,5]. This has led to the development, within the scope of ESA-sponsored initiatives, of the AIR (ARINC 653 in Space RTOS) architecture. The AIR architecture provides time and space partitioning in conformity with the defined in the ARINC 653 specification [3]. AIR preserves the hardware and real-time operating system (RTOS) independence defined within the scope of ARINC 653, while foreseeing the use of different RTOS through the partitions [6,7,8].

However, porting general-purpose applications to one of the RTOSs one might be using can be a morose task, and certainly not an error-free one [9]. Furthermore, certain hardware interfaces may be necessary that are not supported by the given RTOS. This also applies to the aerospace applications that the AIR architecture targets; an example is a space probe for planetary observation, within which a hardware interface with a camera is needed, and whose pictures need to go through some post-processing by a widely available application that has not been ported to the RTOS. Thus we have preliminarily evaluated the integration of generic operating systems, like (embedded) Linux [10]. The execution of non-real-time Linux processes alongside real-time tasks has been studied and implemented before. Examples include RTLinux [11] and xLuna [12], where the non-real-time Linux processes are only scheduled and dispatched when there is no real-time task ready to execute. The approach on the AIR architecture differs in that the non-real-time Linux partition has a guaranteed execution time window in the cyclic, fixed scheduling of partitions.

In this paper, we describe the recent improvements and current efforts on the AIR architecture, towards the flexible integration of both real-time operating systems and generic non-real-time operating systems in partitions. This paper is structured as follows.

In Section 2, we describe the current state of the art concerning the AIR Technology, so as to provide a solid background. Then, in Section 3, we describe in detail the characteristics and advantages of a recently introduced component, the AIR Partition

OS Adaptation Layer (PAL). This component benefits architectural features and the engineering of AIR components (thus adding flexibility to the process of supporting new partition operating systems) and promotes separation of concerns (to optimize development processes at its various stages). In Section 4, we expose the current efforts on exploiting the flexible integration of partition operating systems, to add support for the Linux operating system. Finally, in Section 5, we draw concluding remarks, and lay the foundations for future developments.

2 AIR Technology overview

The AIR activities span over two projects. The first resulted in the development of a proof of concept and a demonstration of feasibility of use [6,7]. The second, AIR-II, which is still in progress, aims evolving towards an industrial product definition by improving and completing the key ideas identified [8].

2.1 System architecture

The fundamental idea in the definition of the AIR architecture is a simple solution for providing the ARINC 653 functionality missing in off-the-shelf (real-time) operating system kernels, encapsulating those functions in special-purpose additional components with well-defined interfaces, as illustrated in the diagram of Fig. 1. In essence, the AIR architecture preserves the hardware and operating system independence defined in the ARINC 653 specification [3]. Applications may use a strict ARINC 653 service interface or, in the case of system partitions, may bypass this standard interface and use partition operating system kernel specific functions, as illustrated in Fig. 1. The existence of system partitions with the possibility of bypassing the APEX interface is a requirement of the ARINC 653 specification. It should noted though that these partitions will typically run system administration and management functions, performed by applications which will be subject to due increased verification efforts.

The following fundamental components have been defined within the AIR system architecture. We will now explain them with as much detail as needed for the understanding of the issues at hand in this paper. More detailed descriptions can be found in previous publications [7,8].

2.2 AIR Partition Management Kernel (PMK)

The AIR Partition Management kernel (PMK), is a simple micro-kernel that efficiently handles partition scheduling and dispatching, thus playing a lead role in securing robust temporal segregation. A two-level hierarchical scheduling [13] scheme is used: partitions are scheduled deterministically at PMK level by a fixed cyclic scheduler; scheduling of the application processes inside each partition is normally handled by the native Partition Operating System (POS) scheduler. RTOS kernels typically offer a preemptive, priority-based process scheduler. At the AIR PMK level, the Partition Scheduler checks at each system clock tick whether a partition preemption is to occur; if it is so, the AIR PMK Partition Dispatcher has to perform the partitions' context switch. The

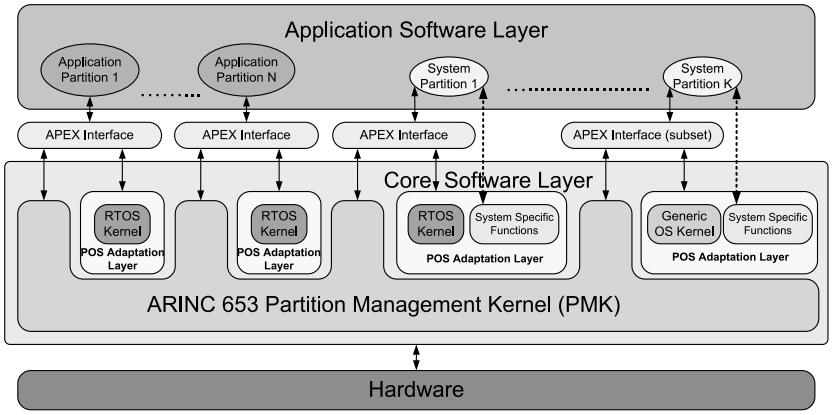


Fig. 1. AIR architecture overview

native POS process scheduler of the heir partition is then notified of the amount of clock ticks elapsed since it was last preempted, thus adjusting the heir partition system time to a common partition-wise time referential.

The design of AIR PMK also incorporates enhanced mechanisms to ensure temporal segregation, like mode-based schedules and process deadline violation monitoring [8].

2.3 Flexible Portable APEX Interface

The APEX Interface implements a set of services defined in the ARINC 653 specification. For generic operating systems (e.g. embedded Linux) only a subset of the APEX standard primitives is needed, primarily for management and monitoring purposes [10]. The APEX design and implementation of the APEX may benefit from the availability of functions related to the recently introduced AIR POS Adaptation Layer (PAL) [8], also detailed in Section 3.

2.4 AIR Health Monitoring (HM)

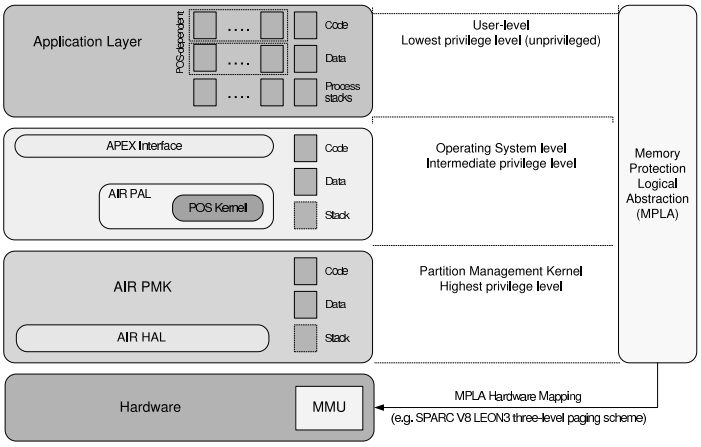
The AIR Health Monitor is responsible for handling hardware and software errors (like deadlines missed, memory protection violations, bounds violation or hardware failures). The aim is to isolate errors within its domain of occurrence: process level errors will cause an application error handler to be invoked, while partition level errors trigger a response action defined in a configuration table. Errors detected at system level may lead the entire system to be stopped or reinitialized [8].

2.5 AIR Space Partitioning and Operating System Integration

The robust partitioning approach defined in the AIR architecture implies the spatial separation of the different (real-time and non real-time) operating systems and its applications in integrity and criticality containers, defined by partitions. Partitions encapsulate

the addressing spaces of the contained POS and applications. No component integrated in a given partition can directly access the addressing space of other partitions, thus guaranteeing that partitions do not interfere with each other [7,8].

A highly modular design approach has been followed in the support of AIR spatial partitioning. Spatial partitioning requirements, specified in ARINC 653 configuration files with the assistance of development tools support, are described in run-time through a high-level processor independent abstraction layer [8]. A set of descriptors is provided per partition, primarily corresponding to the several levels of execution of an activity (e.g. application, POS kernel and AIR PMK) and to its different memory blocks (e.g. code, data and stack), as illustrated in the diagram of Fig. 2.



Memory Management Unit (MMU) Mapping Results				
Processing Platform	***MMU Address Translation Model***	***Primary MMU Descriptors per Partition***	***Number of Partitions***	***Partition Size***
IA-32	segmentation	1	variable	variable
	paging	1	1024	4 MiB
SPARC V8	paging	1	256	16 MiB

Fig. 2. AIR Spatial Partitioning and Operating System Integration

In the AIR architecture the definition of the high-level abstract spatial partitioning takes into account the semantics expected by user-level application programming. At each partition, the application environment inherits the execution model of the corresponding POS and/or its language run-time environment. This is true for system partitions and may be applied also to application partitions, using only the standard APEX interface.

The high-level abstract spatial partitioning description needs to be mapped in run-time to the specific processor memory protection mechanisms, possibly exploiting the

availability of high-level logical address translation schemes, as provided by memory management unit (MMU).

The mapping into MMU specific mechanisms depends on the resources available on each processing platform foreseen for AIR applications. The most versatile mapping assumes the use of a memory segmentation model, such as it exists in the Intel IA-32 architecture, where a one-to-one mapping between the high-level abstract spatial partitioning description and low-level memory management descriptors is possible.

A one-to-one mapping is not possible if a paging translation model is being used in the MMU since a memory descriptor is required by page frame. This also implies some restrictions with respect to the number and size of partitions, as illustrated by the data inscribed in Fig. 2.[1] An optimal design approach is assumed, where the action of changing the status of a partition (active/inactive) requires no more than the update of a single primary MMU descriptor per partition. The data inscribed in Fig. 2, for IA-32 and SPARC V8 RISC processing architectures, is in conformity with the requirements found in typical avionics and aerospace applications.

Mapping of high-level abstract partitioning also includes the management of privilege levels: only the AIR PMK is executed in privileged mode (cf. Fig. 2). The lack of multiple protection rings, such as it exists in the Intel IA-32 processor architecture, may be mitigated in the SPARC V8 architecture by granting access to a given level only during the execution of services belonging to that level (or lower ones). This may be achieved by activating the corresponding memory protection descriptors upon call of a service primitive, and deactivating them when service execution ends.

The provision of these mapping functions is under the scope of overall partition management as provided by the APEX layer and by some specific AIR PMK components. For example, the mapping into processor specific descriptors needs to be updated in run-time when a partition switch occurs. This has to be coordinated by AIR PMK specific components, in this case by the AIR PMK Partition Dispatcher.

3 The AIR POS Adaptation Layer (PAL)

The AIR architecture allows operating systems integration without any fundamental change to a given POS Kernel. In essence, only the OS initialization process and the system clock handler need to be adapted. A generic approach and a uniform methodology have been adopted for the integration of both real-time and generic operating systems [15]. The adopted solution wraps the POS and, if applicable, the system specific functions, through the use of the AIR POS Adaptation Layer (PAL), facilitating the integration at the low- and at high-level domains, i.e. with respect to the AIR PMK and APEX components.

The AIR PAL was added to the original AIR architecture [7] as a means to a truly POS-independent AIR PMK, but its benefits go beyond that. The improvements enabled by the AIR PAL go in three ways: **(i)** architectural properties; **(ii)** engineering of AIR components, and; **(iii)** leaner development processes, stemming from separation of

[1] It is worth mentioning that this paper follows a notation in conformity with the IEC 60027-2 standard in respect to the usage of prefixes for binary multiples [14].

concerns. We will now briefly elaborate on these three direction of benefits, which are more thoroughly described and illustrated in [15].

3.1 Architectural properties

By wrapping each partition's operating system kernel inside an adaptation layer (the AIR PAL), the AIR PMK can act upon the POS in a way that is agnostic of the latter, when necessary. Upon the need to add support to a different POS, the AIR PMK remains unaltered, with support being coded by developing an adequate PAL. This way, previous or ongoing verification, validation and/or certification efforts on the AIR PMK are not hindered. The AIR PAL also benefits the design of other AIR components, such as the Portable APEX and the AIR Health Monitoring (HM) [15].

3.2 Component engineering

Besides consolidating the properties of the AIR architecture and its components, the AIR PAL can also make up for some non-optimal or inappropriate behavior of the native POS implementation of some function. Also, by providing these surrogate functions — intended to be called in spite of the native ones — instead of creating patches to be applied to the POS's source code, we extend the lifetime of the support to a given POS through more subsequent versions of the latter. The reason for this is that the patches' mapping onto their target relies on source code file names and line numbers, whereas the AIR PAL relies on function prototypes and behaviors.

The improvements on architectural properties and component engineering are closely related, and constitute the basis of a flexible integration of partition operating systems.

3.3 Separation of concerns

Another reason against patching the POS so as to obtain the integration and intended behavior for running on the AIR architecture is that it would break the desired separation of concerns, thus undermining an otherwise streamlined development process. Application developers should not be concerned with how the underlying POS is adapted to the AIR architecture, and neither should support for new POSs divert the AIR PMK maintainers' focus from what should be their main concerns — the robust temporal and spatial partitioning properties of AIR.

By consolidating the separation of concerns in the AIR architecture, the development workflow can rely much more on reusable components. This leads to leaner software [10] development processes, with less overhead spent on interactions between different stakeholders (partition application developers, system integrators, etc.).

4 Integration of generic non-real-time operating systems

In [10], we presented the problem of integrating generic operating systems onto the AIR architecture, thus tackling the issue of functionally porting general-purpose applications to an environment provided by real-time operating systems [9].

4.1 Usefulness of Linux-based partitions

One solution for avoiding the effort of porting general-purpose applications to real-time partitions is to have them running on their native operating system.

We have looked into Linux as a candidate for a generic non-real-time partition OS in AIR and shown the development and evaluation of a fully functional operating system, based on the Linux kernel. The integration of Linux makes available to AIR applications a wide range of utilities, tools, language interpreters (Python, Perl, Ruby, tcl, etc.), and device drivers. This specific facilities no longer need to be ported to the RTOS to construct AIR applications. Should it be a requirement, the access to those tools can be supported by AIR inter-partition communication facilities [10].

4.2 Embedded Linux

Given the coexistence of the multiple POSs in the system, which in the absence of persistent storage (e.g. hard disk drive) will be resident in memory during the entire platform execution time, it is particularly important to keep the POSs to a minimum size. Thus, it makes perfect sense using techniques and methodologies aimed at systems with scarce resources — embedded systems.

The embedded variant of Linux which was described and evaluated in [10] was developed around three pillars of optimization: kernel configuration, system library, and utilities/tools. Regarding kernel configuration, size optimizations consisted of selecting only a relevant set of features, and providing those as built-in in the kernel, rather than as loadable modules. The system library used in most typical Linux distributions, the GNU C library (glibc) was replaced by uClibc, more appropriate for systems with scarce resources; uClibc developers accomplished this by reimplementing functionalities with size optimizations in mind, and by modularizing some of them, (allowing the configuration of the uClibc library and its adaptation to the requirements of the target system). Finally, common utilities and tools, usually provided as standalone executables, are provided through a utility called BusyBox, which also provides optimized implementations and allows both selection and fine-tuning of the utilities to include in one single executable file.

To aid building the cross compilation toolchain and the final target system image, we used Buildroot, which also allows (through its simple configuration tool) including extra functionalities as standalone executables, such as a different system shell, or support for interpreted/scripting languages. Figure 3 compares the obtained embedded variant of Linux with a typical desktop distribution.

Sizes illustrated for the system library and utilities/tools account for identical sets of features on both sides. Regarding the Linux kernel, a typical Linux distribution ships a set of loadable kernel modules that can amount to 50 MiB, which were not accounted for in the chart to allow for a fairer comparison.

Since the AIR architecture, when running on a SPARC architecture, will allow for 16 MiB per partition, the obtained size of about 1.5 MiB is very promising, and is also closer to the typical size of space applications.

Kernel		System library		System tools		Total	
Generic	***Embedded***	***glibc***	***uClibc***	***Generic***	***BusyBox***	***Generic***	***Embedded***
2150 KiB	830 KiB	2474 KiB	368 KiB	1932 KiB	363 KiB	6556 KiB[a]	1561 KiB

[a] Plus a set of modules amounting to 50 MiB

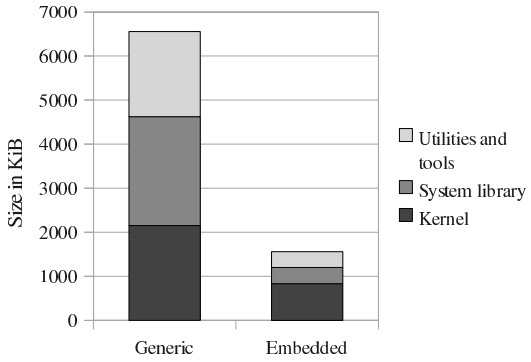


Fig. 3. Overall size comparison between an Embedded Linux and a typical Linux distribution

4.3 Integration issues

Issues being currently researched regarding the integration of Linux as partition OS focus on guaranteeing that it does not contaminate the robust temporal and spatial partitioning of the AIR architecture. Temporal partitioning is ensured, as standard, by the cyclic fixed scheduling of partitions, provided that the Linux partition can not disable or divert interrupts at the hardware (processor) level. We will want the Linux kernel to be notified of clock ticks, like other partition operating systems, only when its partition is active. Thus, interrupts will be totally controlled and handled by the AIR PMK, bypassing the Linux interrupt infrastructure [16].

To guarantee this, and since most processor architectures are not fully virtualizable (i.e., not all sensitive instructions are also privileged instructions), we can not merely run Linux in an unprivileged mode (usermode) and rely on having sensitive instructions generate a trap [17,18]. A good candidate to solve this issue is the employment of paravirtualization [19].

4.4 Paravirtualization in the Linux kernel

The paravirt-ops paravirtualization interface, which enables multiple hypervisors to hook directly into the Linux kernel, has been merged into the main Linux kernel starting with version 2.6.21, along with the support for VMWare's Virtual Machine Interface (VMI). VMI is the open specification of an interface for the paravirtualized guest OS kernel to communicate with the hypervisor [20], which takes advantage of hooks onto the paravirt-ops interface. Many popular GNU/Linux distributions shipping with Linux 2.6.21 have the paravirt-ops and VMI configuration options enabled; this means that the

same kernel will run both on native hardware and on top of a VMI-enabled hypervisor without requiring recompilation (with negligible performance overhead when running on native hardware [21]).

Figure 4 illustrates the process in which a VMI-enabled Linux kernel is booted, and either runs natively or on top of a hypervisor. Early during the boot process, the VMI initialization code probes for a ROM module through which the hypervisor's VMI layer is to be published to the paravirtualized operating system. If such a module is found, the VMI initialization code dynamically patches the kernel, so as to inject the necessary calls to the hypervisor's VMI layer; if not, the kernel continues to run as normal, natively on top of the hardware [20].

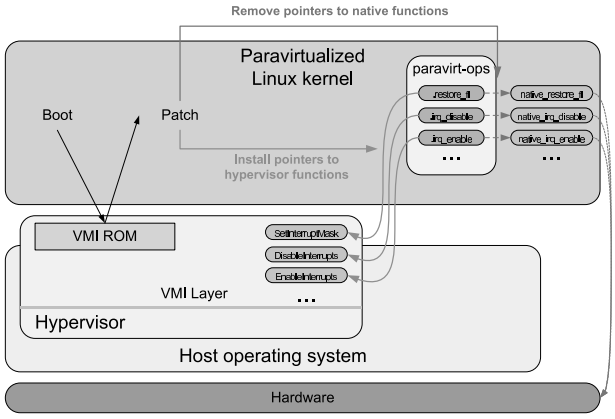


Fig. 4. Boot process of a paravirtualized Linux kernel on top of a VMI-compliant hypervisor

4.5 AIR Linux partition: AIR PAL design and integration

When transposing this to the reality of the AIR architecture, the AIR PAL will provide the relevant functions of the VMI layer to the partition operating system, interacting with AIR PMK when required. Examples of the VMI functions to be provided by the AIR PAL include virtualization of: **(i)** interrupt management; **(ii)** input/output (I/O) calls; **(iii)** memory and I/O space protection mechanisms; **(iv)** privilege level management. This integration is illustrated in Fig. 5.

4.6 AIR application platforms

The space applications to which the AIR technology is to be applied typically employ SPARC-V8 RISC processors, like LEON 2 and LEON 3, so the concepts of paravirt-ops and VMI, which are Intel IA-32 and Intel 64-centric by design, have to be transposed to the reality of this architecture. The current state of the art is nevertheless interesting for

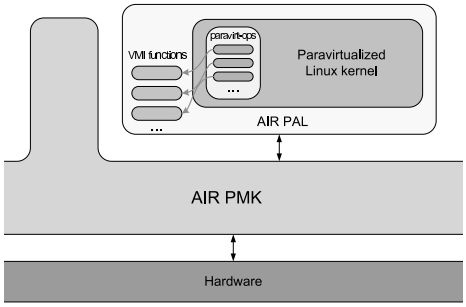


Fig. 5. Concepts of paravirtualization in the AIR architecture

proof of concept prototyping purposes, and to apply to ground-segment applications, where the Intel architectures are present. As of Linux kernel 2.6.30, paravirt-ops and VMI support is implemented for both Intel IA-32 and Intel 64 architectures.

5 Conclusions and future work

In this paper, we have focused on recent and current developments performed on the ARINC 653-based AIR Technology, in order to make the process of adding support to new operating systems, including generic non-real-time operating systems, more flexible. Having described the essentials of the AIR architectural components, we further detailed the latest space segregation results and the recently introduced AIR Partition OS Adaptation Layer (PAL), crucial for the provision of a homogeneous and flexible operating system integration process, which brings stronger architectural properties, benefits the engineering of the nuclear components of the AIR architecture, and improves the development process in its various stages, by promoting separation of concerns.

We also describe the current efforts of integrating Linux as a partition operating systems, focusing on the concepts associated with the paravirtualization interface currently provided by the Linux kernel (paravirt-ops).

There are still challenges open to future developments, both at architectural level and in the provisioning of adequate tools to build ARINC 653-based systems and applications. At the architectural level, future work includes consolidating the application of the concepts of paravirt-ops to the integration of Linux. A first approach will be based on the Intel IA-32 architecture, for which the paravirt-ops interface is already implemented; subsequently, the idea should be ported to the SPARC architecture, namely the LEON processors, employed in space missions. Work concerning the provisioning of adequate tools will include tools for developers and integrators, combining the analysis of the mutual impact between partition- and process-level scheduling with the automated generation of partition scheduling tables.

References

1. Watkins, C., Walter, R.: Transitioning from federated avionics architectures to Integrated Modular Avionics. In: Digital Avionics Systems Conference, 2007. DASC '07. IEEE/AIAA 26th. (October 2007) 2.A.1–1–2.A.1–10
2. Airlines Electronic Engineering Committee (AEEC): Design Guidance for Integrated Modular Avionics, ARINC Specification 651 (1991)
3. Airlines Electronic Engineering Committee (AEEC): Avionics Application Software Standard Interface, ARINC Specification 653-2 Part 1 (Required Services) (March 2006)
4. Terraillon, J.L., Hjortnaes, K.: Technical note on on-board software. European Space Technology Harmonisation, Technical Dossier on Mapping, TOSE-2-DOS-1, ESA (February 2003)
5. TSP Working Group: Time and space partitioning for space application (presentation). In: ESA Workshop on Avionics Data, Control and Software Systems (ADCSS). (October 2008)
6. Diniz, N., Rufino, J.: ARINC 653 in space. In: Proceedings of the DASIA 2005 "DAta Systems In Aerospace" Conference, EUROSPACE (June 2005)
7. Rufino, J., Filipe, S., Coutinho, M., Santos, S., Windsor, J.: ARINC 653 interface in RTEMS. In: Proceedings of Data Systems in Aerospace (DASIA'07). (June 2007)
8. 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 (May 2009)
9. Kinnan, L., Wlad, J., Rogers, P.: Porting applications to an ARINC 653 compliant IMA platform using VxWorks as an example. In: Digital Avionics Systems Conference, 2004. DASC 04. The 23rd. Volume 2. (October 2004) 10.B.1–10.1–8 Vol.2
10. Craveiro, J., Rufino, J., Almeida, C., Covelo, R., Venda, P.: Embedded Linux in a partitioned architecture for aerospace applications. In: Proceedings of the 7th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'09). (May 2009) 132–138
11. Yodaiken, V., Barabanov, M.: A real-time Linux. In: Proc. Linux Applications Development and Deployment Conference (USELINUX). (January 1997)
12. Braga, P., Henriques, L., Carvalho, B., Chevalley, P., Zulianello, M.: xLuna - demonstrator on ESA Mars Rover. In: Proc. of DASIA 2008 — DAta Systems In Aerospace. (May 2008)
13. Audsley, N., Wellings, A.: Analysing APEX applications. In: 17th IEEE Real-Time Systems Symposium. (Dec 1996) 39–44
14. International Electrotechnical Commission (IEC): IEC 60027-2: Letter symbols to be used in electrical technology – Part 2: telecommunications and electronics (August 2005)
15. Craveiro, J., Rufino, J., Schoofs, T., Windsor, J.: Robustness, flexibility and separation of concerns in ARINC 653-based aerospace systems. AIR-II Technical Report RT-09-02 (2009) Submission to IEEE Embedded Systems Letters (ESL).
16. Bovet, D., Cesati, M.: Understanding the Linux Kernel. 3rd edn. O'Reilly Media, Inc. (August 2008)
17. Popek, G.J., Goldberg, R.P.: Formal requirements for virtualizable third generation architectures. Commun. ACM **17**(7) (1974) 412–421
18. Smith, J.E., Nair, R.: Virtual Machines: Versatile Platforms for Systems and Processes. Morgan Kaufmann Publishers (2005)
19. Whitaker, A., Shaw, M., Gribble, S.D.: Denali: Lightweight virtual machines for distributed and networked applications. In: Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation. (June 2002) 195–209
20. Amsden, Z., Arai, D., Hecht, D., Holler, A., Subrahmanyam, P.: VMI: An interface for paravirtualization. In: Proceedings of the Linux Symposium. (July 2006) 363–378
21. VMware, Inc.: Native performance: paravirt vs non-paravirt kernel (May 2009) http://www.vmware.com/interfaces/paravirtualization/performance.html.

Verificação de Modelos de Programas HTL

Joel Silva Carvalho e Simão Melo de Sousa

REliablE And SEcure Computation Group
Departamento de Informática da
Universidade da Beira Interior,
Portugal

Resumo Neste artigo apresenta-se uma *Tool-Chain* para extensão da verificação temporal de programas HTL. Neste processo foi desenvolvida uma ferramenta de tradução automatizada designada por HTL2XTA. Partindo da especificação de um programa HTL, esta ferramenta constrói um modelo Uppaal e um conjunto de propriedades desse modelo. As propriedades inferidas do modelo baseiam-se em propriedades conhecidas do HTL, no entanto é possível acrescentar a essa especificação propriedades que se correlacionem com os requisitos temporais previamente definidos.
This paper introduces a tool-chain that extends the verification of HTL programs. This tool-chain is based on an automated translation tool, called HTL2XTA. This translator builds, from a HTL program, an Uppaal and infers a set of properties that the model (and thus, the source code) should meet. Such properties state the compliance of the model with temporal constraints that can be deduced from HTL source code. In addition to these automatically inferred properties, the Uppaal model can also be completed by any other property that the user may consider.

1 Contexto

Novas exigências surgem com a evolução dos sistemas computacionais. De facto, a capacidade de processamento, por si só, já não é suficiente para o preenchimento de todos os requisitos industriais. Nos sistemas críticos a segurança e a fiabilidade são os aspectos fundamentais[1]. Apesar de ser importante, não basta reunir condições técnicas para executar um dado conjunto de tarefas num sistema, é preciso que o sistema (como um todo) execute correctamente essas tarefas.

Este artigo resulta do estudo da fiabilidade de um subconjunto de sistemas computacionais, mais precisamente os Sistemas de Tempo Real Críticos[2][3]. Comparativamente com os sistemas tradicionais, os sistemas de tempo real acrescentam, à questão da fiabilidade, a necessidade intrínseca de garantir que as tarefas que compõem tais sistemas são executadas individualmente num intervalo de tempo bem determinado. Para este tipo de sistemas, não se conseguir

[1] Este trabalho é parcialmente suportado pelo projecto RESCUE (PTDC/EIA/65862/2006) financiado pela FCT (Fundação para a Ciência e a Tecnologia).

finalizar uma tarefa, no tempo que é devido, corresponde sumariamente a uma falha do sistema.

1.1 Uppaal

O Uppaal[4] foi desenvolvido pelas universidades de **Upp**sala e de **Aal**borg, e consiste numa aplicação de modelação (com redes de autómatos temporizados[5]), simulação e verificação (com um subconjunto da lógica TCTL[4]) de sistemas de tempo real. Uma vez que o motor do verificador de modelos[6] é independente da interface gráfica é possível verificar um modelo tendo apenas a especificação textual do mesmo. A especificação do modelo pode ser feita no formato *ta*, *xta* ou *xml*, e a especificação das propriedades no formato *q*. Esta abordagem foi utilizada na *Tool-Chain* para permitir verificação sem necessitar recorrer à interface gráfica do Uppaal.

1.2 HTL

O HTL (Hierarchical Timing Language)[7][8][9] é uma linguagem de coordenação[10] hierárquica para sistemas de tempo real críticos, com tarefas periódicas, que permite verificação da *time-safety* na vertente do escalonamento. As linguagens de coordenação tem por principal objectivo a combinação e/ou manipulação de linguagens existentes. Estas usufruem das propriedades desejadas de uma ou diversas linguagens servindo de intermediário. Assim, o princípio de base é que, num sistema crítico que contemple uma camada HTL, esta sirva de especificação do comportamento temporal das funções definidas em C/C++. A descrição temporal é separada da descrição funcional das tarefas que compõem o sistema.

Na base do HTL está uma abstração, que separa a execução física das tarefas da sua execução lógica, designada por LET (Logical Execution Time). Sumariamente o LET considera um intervalo de tempo lógico no qual a tarefa pode ser executado independentemente da forma como o sistema operativo distribui os recursos para essa tarefa. O LET de uma tarefa só é iniciado após a última leitura de variável e é finalizado antes da primeira escrita de variável. Esta abstração tem um papel fundamental no esquema de verificação descrito adiante.

1.3 Motivação

Do estudo feito aos mecanismos de verificação, aplicados nos sistemas de tempo real críticos, constatou-se que existem linguagens derivadas do Giotto[11] capazes de verificar estaticamente o escalonamento de programas. Este tipo de linguagens, apesar de ser académico, aufere um conjunto de propriedades interessantes e distintas das ferramentas mais industrializadas. Essas características são o reaproveitamento eficiente do código, a facilidade teórica de adaptação de um mesmo programa a diversas plataformas, a construção dos programas por hierarquias, a possível utilização de todas as características da linguagem funcional sem qualquer limitação, etc. O nosso interesse recaiu para a mais recente

das linguagens derivadas do Giotto, o HTL que culminou com a publicação de duas teses de doutoramento [7][9] no decorrer de 2008, uma vez que a mesma usufrui das diversas evoluções do Giotto. No entanto, esta linguagem ainda requer algum desenvolvimento para que a sua verificação seja mais completa e considerada suficiente no meio industrial.

Tendo em atenção este aspecto, constatou-se ainda que a verificação temporal do HTL podia ser complementada com verificação de modelos[6]. O tipo de verificação do Uppaal complementa muito bem a análise estática realizada pelo compilador HTL. Enquanto no HTL é feita uma análise de escalonamento e é garantido que o sistema ao ser executado cumpre com esse requisito, o Uppaal permite fazer uma análise temporal sobre o comportamento das tarefas (eventos). Se nos requisitos do sistema está especificado que a situação A não pode ser verificada em simultâneo com a situação B, e sabendo quais as tarefas que implementam essas situações, então o Uppaal pode verificar esta propriedade no modelo do sistema.

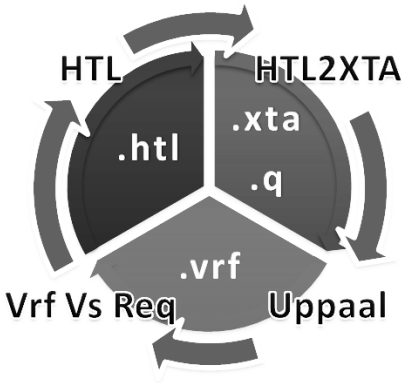
Inspirado em [12] e de forma análoga a [13], mas recorrendo a uma abstração diferente e executando o processo de tradução de forma automatiza, o tradutor que se apresenta no artigo constrói modelos com base em programas HTL e especifica propriedades esperadas sobre os modelos. Após serem verificadas pelo verificador de modelos Uppaal[4], as propriedades permitem estabelecer concordância entre os requisitos temporais e o que foi realmente programado.

2 HTL2XTA Tool-Chain

O tradutor (HTL2XTA) enquadra-se numa *Tool-Chain* (figura 1) delineada com o objectivo de estender a verificação de programas HTL. O HTL2XTA recebe uma especificação HTL (ficheiro .htl) e devolve o respectivo modelo (ficheiro .xta) juntamente com as propriedades automaticamente inferidas (ficheiro .q). É possível, com base nestes ficheiros, utilizar a interface gráfica do Uppaal ou o motor do verificador de modelos (verifyta) para apurar se as propriedades são ou não satisfeitas. De modo a facilitar este processo, foram criados scripts para executar o verificador de modelos gerando um relatório (ficheiro .vrf) por cada modelo verificado. Com este relatório e com a respectiva especificação de propriedades é possível saber se parte dos requisitos temporais são ou não cumpridos.

Para completar a verificação dos requisitos temporais basta acrescentar à especificação de propriedades, automaticamente gerada, novas fórmulas que verifiquem os requisitos que não foram automaticamente contemplados. Propriedades mais interessantes como, 'A tarefa X nunca pode ocorrer em simultâneo com a tarefa Y', ou 'Se a tarefa X ocorrer, passado T unidades de tempo a tarefa Y tem de ocorrer' não são automaticamente geradas. É preciso estudar os requisitos temporais e encontrar correspondências com as tarefas que implementam essas situações de modo a poder especificar propriedades sobre esses requisitos.

Figura 1. HTL2XTA Tool-Chain



2.1 Tradução do Modelo

Sendo o tradutor o meio para atingir o principal objectivo da *Tool-Chain* (a verificação temporal do sistema), decidiu-se evitar a construção de um esquema de tradução que produzisse modelos muito complexos. A razão essencial prende-se com uma limitação clássica da verificação de modelos, nomeadamente o problema da explosão do espaço de estados[6], que não consegue verificar modelos de complexidade alta[1]. Uma vez que o HTL já faz uma análise de escalonamento, a abstração utilizada ignorou por completo a execução física das tarefas. Ao contrário de [13], decidiu-se que a abstração a ter em consideração não devia, nem podia, considerar a execução física das tarefas. Como não interessa verificar o escalonamento das tarefas, o LET do HTL é mais que suficiente para verificar as restantes propriedades temporais.

Fez-se assim corresponder a cada invocação de tarefa um autómato temporizado cujo LET é definido através do cálculo dos portos concretos e dos comunicadores utilizados na sua declaração. O limite inferior do LET corresponde ao momento em que é lida a última variável e o limite superior corresponde ao momento em que é escrita a primeira variável.

Uma vez que cada modo dentro de um módulo representa a execução de um conjunto de tarefas e que cada módulo só pode estar num modo de cada vez, faz-se corresponder a cada módulo um único autómato temporizado. A cada ciclo de execução do autómato, faz-se a sincronização com os autómatos representativos das tarefas invocadas num determinado modo. Na abstração adoptada é ignorado por completo o tipo dos comunicadores bem como o driver de inicialização.

Sumariamente todo o esquema de tradução rege-se por esta abstração, i.e. autómatos temporizados que representam o LET das tarefas e autómatos temporizados que representam cada módulo. O tradutor foi testado com diversos

[1] O tamanho do modelo global dum sistema é exponencial comparativamente ao tamanho das suas componentes. A verificação destes modelos obriga a uma exploração, muitas vezes exaustiva, do modelo global

programas HTL e constatou-se que os modelos de programas de complexidade mais elevada não permitem verificação completa (por exemplo a complexidade de um dos modelos não permitiu a verificação da ausência de bloqueio), apesar da simplicidade da abstração. Todos os casos de complexidade intermédia foram alvo de verificação bem sucedida, mas a quantidade de refinamentos aumenta substancialmente a complexidade do modelo. Para aliviar esta situação, o tradutor permite a construção de modelos tendo em conta os níveis de refinamento desejados.

2.2 Inferência de Propriedades

As propriedades inferidas estão todas relacionadas com características bem definidas do HTL, como os períodos dos modos, o LET de cada tarefa, as invocações de tarefas feitas em cada modo e o refinamento dos programas. A cada uma das características referidas corresponde, quase sempre, mais do que uma propriedade. À semelhança de uma tabela de restreabilidade, cada propriedade é devidamente comentada com, uma descrição textual da característica a verificar, uma referência da posição da respectiva descrição da característica no ficheiro HTL e o resultado booleano pretendido nessa propriedade.

Listing 1.1. Exemplo de propriedades comentadas

```
1  /* Deadlock Free -> true */
2  A[] not deadlock
3  /* P1 mode readWrite period 500 @ Line 19 -> true */
4  A[] sP_3TS_IO.readWrite imply ((not sP_3TS_IO.t>500) && (not sP_3TS_IO.
       t<0))
5  /* P2 mode readWrite period 500 @ Line 19 -> true */
6  sP_3TS_IO.readWrite --> (sP_3TS_IO.Ready && (sP_3TS_IO.t==0 ||
       sP_3TS_IO.t==500))
7  /* P1 Let of t_write = [400;500] @ Line 21 ->true */
8  A[] (IO_readWrite_t_write.Let imply (not IO_readWrite_t_write.tt<400 &&
       not IO_readWrite_t_write.tt>500))
```

As propriedades inferidas podem e devem ser complementadas manualmente com informação extraída dos requisitos temporais estabelecidos. Para tal é preciso ter em consideração a identificação de todos os estados e dos respectivos autómatos temporizados. Considerando um Programa P, um módulo M, um modo m e uma invocação de tarefa t, o autómato do módulo M é identificado por sP_M, o estado que representa a invocação da tarefa é identificado por $sP\_M.m\_t$, o autómato representativo do LET da tarefa (futuramente designado por autómatos de tarefa) é identificado por M_m_t e o estado do próprio LET é identificado por $M\_m\_t.Let$. Associado a cada autómato de um módulo existe ainda um estado representativo da execução de cada modo identificado por $sP\_M.m$, bem como outros estados que não possuem uma relação directa com o HTL. Exemplificando com a listagem 1.1, na linha quatro e seis tem-se que $P = P\_3TS$, $M = IO$, $m = readWrite$ e na linha oito $t = t\_write$.

A especificação de propriedades permite a utilização de diversos relógios presentes no modelo. Cada autómato é constituído de, pelo menos, um relógio local reinicializado numa transição que sai do estado inicial. No caso dos autómatos de módulo, o relógio local é designado por t e é identificado de forma análoga a

um estado desse autómato. No entanto a utilização de um relógio implica que o mesmo seja comparado com uma expressão inteira, por exemplo: $sP\_M.t >= 0$. No caso dos autómatos das tarefas, o relógio local (representativo do período do modo) onde essa tarefa é invocada é designado por tt. Neste tipo de autómatos existe ainda outro relógio local designado por t reinicializado no instante 0 do LET dessa tarefa. Existe um relógio global designado por *global* que, apesar de não ser utilizado em nenhuma propriedade inferida do modelo, pode ser utilizado nas propriedades especificadas manualmente.

3 Algoritmo de Tradução do Modelo

Alguns aspectos da linguagem HTL são puramente ignorados pelo algoritmo do tradutor. Ou porque não acrescentam informação relevante, ou porque não são suficientemente abstractos para o modelo. Uma vez que a AST (Abstract Syntax Tree) do tradutor foi inicialmente pensada para suportar a descrição da totalidade da linguagem HTL, a mesma possui informação que não é analisada ou traduzida pelo algoritmo.

Considera-se a definição da função T, que aceita um programa HTL (de facto a AST do HTL) e que devolve a RAT (Rede de Autómatos Temporizados) correspondente. Esta função é definida naturalmente por recursividade sobre a estrutura da AST da linguagem HTL. Assim passa-se a definir T para cada caso particular da AST em questão. Considera-se ainda a função auxiliar A, que aceita um programa HTL e que devolve informação necessária para construção da RAT.

3.1 Declaração de Comunicadores

Seja (n, dt, pd, p) a declaração de um comunicador, onde n é o nome do comunicador, dt o tipo do comunicador e o driver de inicialização (ct, ci), pd o período do comunicador e p a posição no ficheiro HTL da declaração do comunicador, então $\forall communicator \in Prog, T_{communicator}(n, dt, pd, p) = \emptyset$. Mais uma vez, a aplicação do algoritmo de tradução ignora a declaração dos comunicadores. A declaração do comunicador *com* não tem uma representação directa na abstração adoptada, no entanto a utilização do mesmo é analisada nas invocações de tarefas para determinar o LET, ou seja, $\forall communicator \in Prog$ em que $n = com$ então $A_{communicator}(n, dt, pd, p) = pd$.

3.2 Transposição do LET

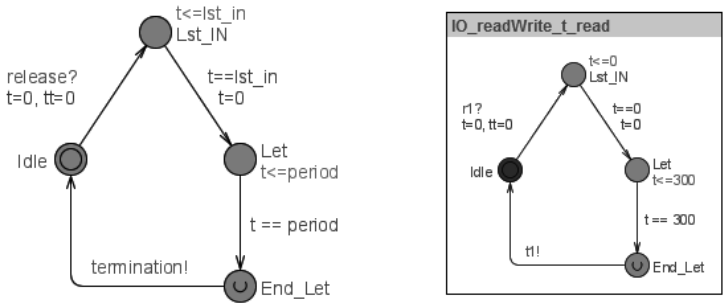
Na base do algoritmo de tradução está uma implementação do LET. Esta implementação é retratada pelos autómatos temporizados *taskTA*, *taskTA_S*, *taskTA_R*, *taskTA_SR*. Existem quatro implementações devido à utilização de portos concretos nas invocações das tarefas. As instanciações do autómato temporizado, $taskTA$ representam invocações de tarefas onde apenas são utilizados comunicadores, $taskTA\_S$ (S de *send*) invocações onde é utilizado um porto concreto

na saída, $taskTA\_R$ (R de *receive*) invocações onde é utilizado um porto concreto na entrada e $taskTA\_SR$ invocações onde é utilizado um porto concreto na entrada e outro na saída.

Invocações de tarefas Considera-se (n, ip, op, s, pos) uma invocação de uma tarefa, onde n é o nome da tarefa a invocar, ip o mapeamento dos portos (variáveis) de entrada, op o mapeamento dos portos (variáveis) de saída, s o nome da tarefa pai e pos a posição no ficheiro HTL da declaração da invocação.

taskTA Seja $Port$ o conjunto de todos os portos concretos, cp um porto concreto e (r, t, p, li) um autómato temporizado $taskTA$ onde, r é uma sincronização urgente de *release*, t uma sincronização urgente de *termination*, p o período do LET da tarefa e li o instante em que a última variável de entrada é lida então $\forall cp \in Port, \forall invoke \in Prog, cp \notin ip, cp \notin op, T_{invoke}(n, ip, op, s, pos) = taskTA(r, t, p, li)$.

Figura 2. Autómato $taskTA$ à esquerda e instanciação à direita



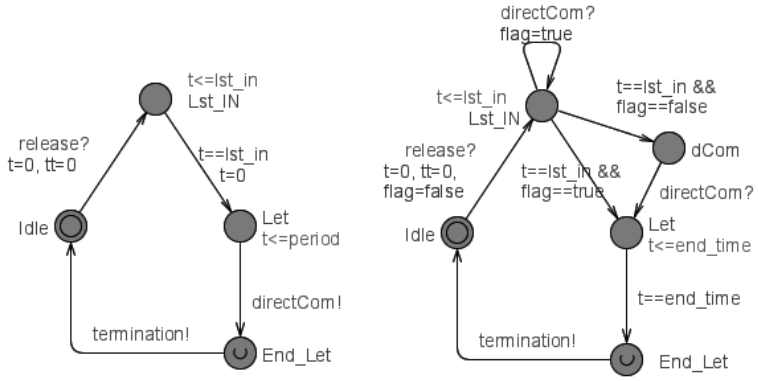
A cada invocação de tarefa, onde não exista nenhum porto concreto, quer nas variáveis de entrada como de saída, corresponde uma instanciação de um autómato $taskTA$ (figura 3.2). Os canais de sincronização urgentes r e t são determinados na declaração do sistema. O nome do canal r de cada instanciação de tarefa é único e produzido de forma enumerada $r1, r2, r3, ...$ O nome do canal t de cada instanciação de tarefa é único, para cada conjunto de autómatos de um modo, e produzido de forma enumerada $t1, t2, t3, ...$

O instante em que a última variável de entrada li é lida determina-se pelo valor máximo da instância de cada comunicador de entrada multiplicado pelo período, no caso de não existir nenhuma variável de entrada então o instante é o zero. O período p do LET é a diferença entre o instante da escrita do primeiro

porto de saída (se não existir nenhum então é o valor do período do respectivo modo) e li.

taskTA_S Seja (r, t, dc, p, li) um autómato temporizado $taskTA\_S$ onde, r é uma sincronização urgente de *release*, t uma sincronização urgente de *termination*, dc uma sincronização urgente de comunicação directa ($directCom$), p o período do LET da tarefa e li o instante em que a última variável de entrada é lida então $\forall cp \in Port, \forall invoke \in Prog, cp \notin ip, cp \in op, T_{invoke}(n, ip, op, s, pos) = taskTA_S(r, t, dc, p, li)$.

Figura 3. Autómato $taskTA\_S$ à esquerda e $taskTA\_R$ à direita



A cada invocação de tarefa, onde exista um porto concreto nas variáveis de saída e nenhum nas de entrada, corresponde uma instanciação de um autómato $taskTA\_S$. Este autómato é muito semelhante ao $taskTA$, introduzindo apenas a questão da sincronização para comunicação directa.

taskTA_R Seja (r, t, dc, p, li) um autómato temporizado $taskTA\_R$ onde, r é uma sincronização urgente de *release*, t uma sincronização urgente de *termination*, dc uma sincronização urgente de comunicação directa ($directCom$), p o período do LET da tarefa e li o instante em que a última variável de entrada é lida então $\forall cp \in Port, \forall invoke \in Prog, cp \in ip, cp \notin op, T_{invoke}(n, ip, op, s, pos) = taskTA\_R(r, t, dc, p, li)$.

A cada invocação de tarefa, onde exista um porto concreto nas variáveis de entrada e nenhum nas de saída, corresponde uma instanciação de um autómato $taskTA\_R$. Este autómato é semelhante ao $taskTA$, necessitando de uma sincronização para comunicação directa.

3.3 Módulos e Modos

Considerando (n, h, mi, bm, pos) um módulo, onde n é o nome do módulo, h é a lista de *hosts*, mi o modo inicial, bmu o corpo do modulo e pos a posição no ficheiro HTL da declaração do módulo. Seja (ref, rl, tl) um autómato temporizado $moduleTA$, onde ref é um canal de sincronização urgente de refinamento (se existir), rl o conjunto dos canais de sincronização urgentes de *release* de todas as invocações de tarefas de um módulo e tl o conjunto dos canais de sincronização urgentes de *termination* de todas as invocações de tarefas de um módulo, então $\forall module \in Prog, T_{module}(n, h, mi, bm, pos) = moduleTA(ref, rl, tl)$.

Seja $(n, p, refP, bmo, pos)$ um modo, onde n é nome do modo, p o período, $refP$ o programa que refina esse modo (caso exista), bmo o corpo do modo e pos a posição no ficheiro HTL da declaração do modo. Seja (e, t) um subconjunto $subModule$ da declaração do autómato temporizado $moduleTA$, onde e é um conjunto de estados (com invariantes) e t um conjunto de transições (com guardas, actualizações e sincronizações) então $\forall mode \in module, \exists subModule \in modulteTA, T_{mode}(n, p, refP, bmo, pos) = subModule(e, t)$.

4 Algoritmo de Tradução das Propriedades

Considera-se a definição da função P, que aceita um programa HTL (de facto a AST do HTL) e que devolve a especificação das propriedades a verificar. Esta função é definida naturalmente por recursividade sobre a estrutura da AST da linguagem HTL. Assim passa-se a definir P para cada caso particular da AST em questão.

4.1 Ausência de Bloqueio

Seja $Prog$ o conjunto de todos os programas e df a descrição da propriedade de ausência de bloqueio, então $P_{Prog} = df$. A aplicação do algoritmo a qualquer programa produz sempre a propriedade de ausência de bloqueio ($A[]$ *not deadlock*).

4.2 Período dos modos

Seja $(n, p, refP, bmo, pos)$ um modo, onde n é nome do modo, p o período, $refP$ o programa que refina esse modo (caso exista), bmo o corpo do modo e pos a posição no ficheiro HTL da declaração do modo. Seja $(p1, p2)$ a especificação das propriedades vm do período de um modo, então
$\forall mode \in Prog, P_{mode}(n, p, refP, bmo, pos) = vm(p1, p2)$.

Seja $moduleTA$ um autómato de módulo e Rat um conjunto de autómatos temporizados, então $\forall mode \in Prog, \exists moduleTA \in Rat,\ p1 = A[]\ moduleTA.n$ *imply* $((not\ moduleTA.t > p)\ \&\&\ (not\ moduleTA.t < 0)),\ p2 = moduleTA.n \rightarrow\ (moduleTA.Ready\ \&\&\ (moduleTA.t == 0\ ||\ moduleTA.t == p)$.

A primeira propriedade $p1$ indica que sempre que o estado de controlo for o estado do modo, isso implica que o relógio local desse autómato de módulo não

seja superior ao período do modo ou inferior a zero. A segunda propriedade $p2$ indica que sempre que é atingido o estado do modo, o estado *Ready* também é atingido e quando isso acontecer o relógio local ou é zero ou é precisamente o valor do período. A combinação destas duas propriedades permite limitar o período do modo ao intervalo $[0, p]$ e ter a garantia que o valor máximo do período é atingido.

4.3 Invocações de tarefas

Seja (n, ip, op, s, pos) uma invocação de tarefa, onde n é o nome da tarefa a invocar, ip o mapeamento dos portos (variáveis) de entrada, op o mapeamento dos portos (variáveis) de saída, s o nome da tarefa pai e pos a posição no ficheiro HTL da declaração da invocação. Seja $(p1, p2)$ a especificação das propriedades vi da invocação de tarefa num modo, então $\forall invoke \in Prog, P_{invoke}(n, ip, op, s, pos) = vi(p1, p2)$.

Seja $taskTA_i$ o autómato da tarefa i, $taskTA$ o conjunto dos autómatos de tarefa, $taskState_i$ o estado da invocação da tarefa i, $modeState$ o estado do modo em que a invocação é feita, $moduleTA$ um autómato de módulo e Rat um conjunto de autómatos temporizados, então $\forall i, \exists moduleTA \in Rat, \exists taskTA_i \in TaskTA, p1 = A[]\ (moduleTA.taskStaste_i\ imply\ (not\ taskTA_i.Idle))\ \&\&\ (moduleTA.Ready\ imply\ taskTA_i.Idle), p2 = A[]\ (taskTA_i.Let\ \&\&\ taskTA.tt! = 0)\ imply\ moduleTA.modeState$.

A propriedade $p1$ indica que para todas as execuções, sempre que o estado de uma invocação é o estado de controlo, isso implica que o respectivo autómato de tarefa não esteja no estado $Idle$ e no respectivo $moduleTA$ quando o estado de controlo é o estado *Ready* isso implica que o autómato de tarefa esteja no estado $Idle$. A segunda propriedade indica que sempre que o estado Let de um autómato de tarefa é o estado de controlo e o relógio local tt é diferente de zero isso implica que a execução do autómato do módulo respectivo esteja no estado representativo do modo onde as tarefas são invocadas.

4.4 LET das tarefas

Seja $(p1, p2, p3)$ a especificação das propriedades $vlet$ da invocação de tarefa num modo, então $\forall invoke \in Prog, P_{invoke}(n, ip, op, s, pos) = vlet(p1, p2, p3)$, $\forall i, \exists moduleTA \in Rat, p1 = A[]\ (taskTA_i.Let\ imply\ (not\ taskTA_i.tt < 0\ \&\&\ not\ taskTA_i.tt > p)), p2 = A <>\ moduleTA.modeState\ imply\ (taskTA_i.Lst\_IN\ \&\&\ taskTA_i.tt == 0), p3 = A <>\ moduleTA.modeState\ imply\ (taskTA_i.Let\ \&\&\ taskTA_i.tt == p)$.

A validação do LET é feita com três propriedades distintas. A propriedade $p1$ indica que sempre que o Let de uma tarefa é atingido, isso implica que o relógio local tt desse autómato de tarefa não seja nem menor que zero nem maior que o período do LET. A propriedade $p2$ indica que sempre que o estado do modo é atingido, inevitavelmente o estado Lst_IN é atingido com o relógio local tt a zero. A propriedade $p3$ indica que sempre que o estado do modo é atingido,

inevitavelmente o estado *Let* é atingido com o relógio *tt* no valor máximo do período da tarefa.

5 Validação Experimental

Utilizando a versão actual do tradutor (v0.4 de 24/04/2009) conseguiu-se gerar, com sucesso, modelos e propriedades para diversos programas HTL apresentados em [7][9]. Na tabela 1 constam algumas informações pertinentes sobre esses resultados. Nomeadamente o número de níveis aplicados na tradução (0=todos, 1=programa principal), o número de linhas do respectivo ficheiro HTL, o número de linhas do ficheiro da especificação do modelo, o número de propriedades especificadas contra o número de propriedades correctamente verificadas bem como o número de estados explorados por cada verificação.

Como seria espectável, sempre que apenas é modelado o programa principal, todos os valores associados ao modelo e à verificação do mesmo tornam-se inferiores. O quanto são inferiores apenas depende do grau de abstração do programa principal. Exceptuando no programa do *Steer By Wire* onde se verificou explosão de estados, foram verificadas todas as propriedades automaticamente produzidas.

Tabela 1. Resultados

Ficheiro	Níveis	HTL	Modelo	Verificações	Estados
3TS-simulink.htl	0	75	263	62/62	8'241
	1	75	199	30/30	684
3TS.htl	0	90	271	72/72	23'383
	1	90	207	40/40	1'216
3TS-FE2.htl	0	134	336	106/106	365'587
	1	134	208	42/42	1'584
3TS-PhD.htl	0	111	329	98/98	214'083
	1	111	201	34/34	1'116
flatten_3TS.htl	0	60	203	31/31	448
steer-by-wire.htl	0	873	1043	617/0	N/A
	1	873	690	394/0	N/A

Não se deve esquecer que o número de linhas de um programa HTL não corresponde ao número de linhas de um programa funcional. Uma vez que a especificação funcional é feita fora do HTL, o facto de um programa ter um número de linhas aparentemente reduzido não implica que se trate de um programa trivial. Por exemplo no caso de estudo do 3TS[7][9] existe algum grau de complexidade (aqui considerado intermédio ou standard) e nenhuma das implementações possui mais de 150 linhas de código HTL. Na realidade estes programas coordenam funções que por si podem ser bastante complexas.

O programa da planta real $3TS - FE2$, por ter mais dois refinamentos do que a versão da planta simulada $3TS - simulink$, tem um aumento considerável

na sua complexidade. Revela-se necessário estudar melhor a relação entre as hierarquias e o aumento de complexidade da verificação do modelo para concluir algo que possa beneficiar o tradutor.

6 Conclusão e Trabalho Futuro

O tradutor HTL2XTA apresentado foi desenvolvido em ocaml, no âmbito de uma dissertação de mestrado, seguindo o modelo de construção de um tradicional compilador, deixando a verificação de tipos ao compilador clássico do HTL. A totalidade da *Tool-Chain* (compilador HTL, tradutor HTL2XTA, Uppaal) funcionam em linux. A linguagem HTL surge num contexto académico e ainda lhe resta completar o desafio da sua transferência de tecnologia para um contexto industrial. Entendemos este nosso trabalho como uma contribuição à resolução deste desafio.

Relativamente às vantagens da verificação proporcionada pela *Tool-Chain HTL2XTA* pode-se dar um dos exemplos testados no caso do 3TS[7][9]. Adulterando a descrição HTL para que o modo $readWrite$ tenha um período de 5000 em vez de 500 isto tem por efeito que o novo executável faça dez vezes mais cálculos com base nas mesmas leituras dos sensores. Em termos de escalonamento a consequência desta alteração é nula na medida que não inviabiliza o escalonamento do programa. Todavia isto infringe os requisitos temporais que exigem que por cada leitura do mesmo sensor seja feita uma actualização do valor do actuador da respectiva bomba de água.

A *Tool-Chain HTL2XTA* não identifica automaticamente esta situação, através da falha da verificação de uma das propriedades, no entanto após comparação dos requisitos temporais com a especificação de cada propriedade gerada é possível constatar a discrepância existente entre os LET's das tarefas e os períodos dos diversos modos. Outra forma de identificar o erro directamente consiste na especificação manual da propriedade apresentada na listagem 1.2. Esta propriedade exige que no instante imediatamente antes da execução do LET da tarefa t_T1 o Let da tarefa t_read esteja em execução. No caso do programa HTL com o período do modo $readWrite$ certo esta propriedade é satisfeita, no caso do período ser maior (independentemente do valor) esta propriedade não vai ser satisfeita.

Listing 1.2. Exemplo de especificação de uma propriedade suplementar

```
1  A[] (T1_m_T1_t_T1.Lst_IN && (T1_m_T1_t_T1.tt>T1_m_T1_t_T1.lst_in-1) &&
       T1_m_T1_t_T1.tt<T1_m_T1_t_T1.lst_in) imply IO_readWrite_t_read.Let
```

Apesar de o tradutor estar numa versão estável e do mesmo conseguir traduzir efectivamente os diversos programas referidos na tabela 1, este não foi alvo de verificação formal. Um dos trabalhos futuros passará, de alguma forma, pela verificação formal da correção do mesmo. No entanto, tendo em consideração que se constatou que esta versão não é capaz de lidar com programas HTL de maior dimensão ainda é cedo para se ter esse esforço.

Outro trabalho futuro contemplará a extensão do próprio HTL com anotações que permitam introduzir regras comportamentais suplementares, como por exemplo o caso dos *switch*. Mas antes disso, deve ser estudada a forma como lidar com essas anotações e o impacto das mesmas no modelo. Por fim pretende-se transpor este modelo, ou um modelo muito similar, para o Ada/SPARK. Isto é, quer-se utilizar os ensinamentos aqui adquiridos para construir um tradutor capaz de abstrair a coordenação das tarefas declaradas nesta linguagem e utilizar o modelo desta coordenação para realizar uma análise temporal estendida.

Referências

1. J. Rushby, "Formal methods and their role in the certification of critical systems," tech. rep., Safety and Reliability of Software Based Systems (Twelfth Annual CSR Workshop, 1995.
2. S.-T. Levi and A. K. Agrawala, *Real-time system design.* New York, NY, USA: McGraw-Hill, Inc., 1990.
3. P. J. Stankovic and A. Stankovic, "Real-time computing," 1992.
4. G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004* (M. Bernardo and F. Corradini, eds.), no. 3185 in LNCS, pp. 200–236, Springer–Verlag, September 2004.
5. J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," 2004.
6. E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model checking.* Cambridge, MA, USA: MIT Press, 1999.
7. A. Ghosal, T. A. Henzinger, D. Iercan, C. Kirsch, and A. L. Sangiovanni-Vincentelli, "Hierarchical timing language," Tech. Rep. UCB/EECS-2006-79, EECS Department, University of California, Berkeley, May 2006.
8. A. Ghosal, *A Hierarchical Coordination Language for Reliable Real-Time Tasks.* PhD thesis, EECS Department, University of California, Berkeley, Jan 2008.
9. D. Iercan, *Contribuitions to the Development of Real-Time Programming Techniques and Technologies.* PhD thesis, EECS Department, University of California, Berkeley, Set 2008.
10. D. Gelernter and N. Carriero, "Coordination languages and their significance," *Commun. ACM*, vol. 35, no. 2, pp. 97–107, 1992.
11. T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *EMSOFT '01: Proceedings of the First International Workshop on Embedded Software*, (London, UK), pp. 166–184, Springer-Verlag, 2001.
12. K. Lundqvist and L. Asplund, "A ravenscar-compliant run-time kernel for safety-critical systems*," *Real-Time Syst.*, vol. 24, no. 1, pp. 29–54, 2003.
13. R. K. Poddar and P. Bhaduri, "Verification of giotto based embedded control systems," *Nordic J. of Computing*, vol. 13, no. 4, pp. 266–293, 2006.

A Fast Convergence Coordination Protocol for Cooperative Open Real-Time Environments

Luís Nogueira, Luís Miguel Pinho

IPP Hurray Research Group
School of Engineering, Polytechnic Institute of Porto, Portugal
{luis,lpinho}@dei.isep.ipp.pt

Abstract. Although a lot of research has been conducted on the support of services' dynamic QoS reconfiguration under different constraints of resource availability, most has been focused on the runtime management of independent QoS attributes. This paper proposes support for adaptive cooperative coalitions of nodes, able to autonomously organise, regulate and optimise themselves without the user's intervention or any other central entity, even when services exhibit unrestricted distributed QoS inter-dependencies among their tasks.

Resumo Apesar da extensa investigação no suporte de uma reconfiguração dinâmica da QoS de serviços sob diversas restrições de disponibilidade de recursos, a maioria tem sido focada na gestão em tempo de execução de atributos de QoS independentes. Este artigo propõe suporte à adaptação de coligações cooperativas de nós, capazes de autonomamente se adaptarem, regularem e optimizarem sem a intervenção do utilizador ou outra qualquer entidade central, mesmo quando os serviços exibem inter-relações de dependência de QoS entre as suas tarefas.

1 Introduction

Adaptive real-time is a relatively new approach to open embedded systems design. It allows an online reaction to load variations, adapting the system to the specific constraints of devices and users, nature of executing tasks and dynamically changing environmental conditions. This ability is essential to efficiently manage the provided Quality-of-Service (QoS) in domains such as telecommunications, consumer electronics, industrial automation, and automotive systems.

Nevertheless, given the heterogeneity of services to be executed, users' quality preferences, underlying operating systems, networks, devices, and the dynamics of their resource usages, developing adaptive distributed cooperative systems presents a number of challenges. These include (i) defining a common understanding of how QoS should be specified; (ii) the provisioning of self-management actions that allow nodes to adapt to observed changes in the environment; (iii) resource management and scheduling strategies; and (iv) the design of an efficient coordination model that regulates individual autonomous adaptive actions. Here, the term *coordinated adaptation* refers to

the ability of a distributed adaptive system to invoke adaptive actions on multiple nodes in a coordinated manner so as to achieve a common goal.

While there has been a great deal of research in the former aspects of adaptation in embedded real-time systems, there has been little work that addresses the specific problem of coordinating individual autonomous adaptations in distributed environments. However, coordination is critical to maintain the correctness of a distributed application during adaptation [1,2] and also very challenging.

In particular, a QoS-aware adaptation in a service's distributed execution can require communication and synchronisation among nodes, used as a building block for a collective adaptation behaviour that emerges from local interactions among nodes. One challenge is controlling this exchange of information in order to achieve a convergence to a globally consistent solution without overflowing nodes with messages.

This paper shows how it is possible to achieve higher levels of self-adaptiveness in systems required to work in open real-time environments through a fast convergence decentralised coordination model. With the increasing size and complexity of open embedded systems the ability to build self-managed distributed systems using centralised coordination models is reaching its limits [3], as solutions they produce require too much global knowledge.

The one-step decentralised coordination model proposed in this paper defines a set of rules through which a coalition of nodes can be reconfigured, imposing restrictions on the way their members can self-adapt in response to changes in the environment. By exchanging feedback on the desired self-adaptive actions, nodes converge towards a global solution, even if that means not supplying their individually best solutions. As a result, each node, although autonomous, is influenced by, and can influence, the behaviour of other nodes in the system.

2 System model

Consider an open distributed system with several heterogeneous nodes, each with its specific set of resources R_i where independently developed services, some of them with real-time execution constraints, can appear while other are being executed, at any time, at any node. Due to these characteristics, resource availability is highly dynamic and unpredictable in advance.

It is assumed that each service S has a set of QoS parameters that can be changed in order to adapt the system's service provisioning to a dynamically changing environment. Each subset of parameters that relates to a single aspect of service quality is named as a *QoS dimension*. Each of these QoS dimensions has different resource requirements for each possible level of service. We make the reasonable assumption that services' execution modes associated with higher QoS levels require higher resource amounts.

There may exist QoS dependencies among two or more of the multiple QoS dimensions of a service S [4]. Given two QoS dimensions, Q_a and Q_b, a QoS dimension, Q_a, is said to be dependent on another dimension Q_b if a change along the dimension Q_b will increase the needed resource demand to achieve the quality level previously achieved along Q_a.

Users provide a single specification of their own range of QoS preferences Q_{Pref} for a complete service S, ranging from a desired QoS level $L_{desired}$ to the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$, without having to understand the individual components that make up the service. As a result, the user is able to express acceptable compromises in the desired QoS and assign utility values to QoS levels. Note that this assignment is decoupled from the process of establishing the supplied service QoS levels themselves and determining the resource requirements for each level.

For some of the system's nodes there may be a constraint on the type and size of services they can execute within the users' acceptable QoS levels. Therefore, the CooperatES framework [5,23] allows a distributed cooperative execution of resource intensive services in order to maximise the users' satisfaction with the obtained QoS. By redistributing the computational load across a set of nodes, a cooperative environment enables the execution of far more complex and resource-demanding services than those that otherwise would be able to be executed on a stand-alone basis.

Nodes dynamically group themselves into a new coalition, allocating resources to each new service and establishing an initial Service Level Agreement (SLA), a service description whose parameters are within the range of the user's desired QoS level $L_{desired}$ and the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$. The coalition is dynamically formed as the set of nodes which maximise the satisfaction of the QoS constraints associated with the new service and minimise the impact on the global QoS caused by the new service's arrival [5].

Nevertheless, short term dynamic environmental changes impose that the promised SLA for a service S_i can never be more than an expectation of a best-effort service quality during long term periods [6]. The system must be adaptive, that is, it must be able to adapt its timing expectations to the current conditions of the environment, possibly sacrificing the quality of other, non-time related, parameters. As such, once a SLA is admitted, it may be downgraded to a lower QoS level in order to accommodate new service requests with a higher utility or (re)upgraded when the needed resources become available.

As such, there will be a set of service's blocks being executed by a group of nodes, resulting from partitioning the resource intensive service S. Due to the existence of QoS dependencies, there are groups of service's tasks that must be executed in the same node. We refer to such groups of tasks as work units. A work unit $w_i = \tau_1, \tau_2, \ldots, \tau_n$ is a set of one or more tasks τ_j of a service S that must be executed in the same node due to local QoS dependencies.

Let $W = \{w_1, \ldots, w_n\}$ be the finite set of work units of a service S. We represent the set of inter-dependencies among work units $w_i \in W$ as a connected graph $\mathcal{G}_{\mathcal{W}} = (\mathcal{V}_W, \mathcal{E}_W)$, on top of the service's distribution graph, where each vertex $v_i \in \mathcal{V}_{\mathcal{W}}$ represents a work unit w_i and a directed edge $e_i \in \mathcal{E}_{\mathcal{W}}$ from w_j to w_k indicates that w_k is functionally dependent on w_j. Within $\mathcal{G}_{\mathcal{W}} = (\mathcal{V}_W, \mathcal{E}_W)$, we call *cut vertex* to a node $n_i \in \mathcal{V}_{\mathcal{W}}$, if the removal of that node divides $\mathcal{G}_W$ in two separated graphs.

We assume that each work unit $w_i \in W$ is being executed at its current QoS level Q^i_{val} at a node n_i, from a set of predefined QoS levels $\{Q_0, \ldots, Q_n\}$, ranging from the user's desired QoS level $L_{desired}$ to the maximum tolerable service degradation,

specified by the minimum acceptable QoS level $L_{minimum}$. This relation is represented by a triple (n_i, w_i, Q^i_{val}). Furthermore, for a given work unit $w_i \in W$, each node n_i knows the set $I_{w_i} = \{(n_j, w_j, Q^j_{val}), \ldots, (n_k, w_k, Q^k_{val})\}$, describing the quality of all the inputs related to work unit w_i coming from its adjacent nodes in $\mathcal{G}_{\mathcal{W}}$.

3 Autonomous cooperative systems

One of the key ideas of the CooperatES framework is that each coalition member should be able to take the initiative and to decide when and how to adapt to changes in the environment. However, whenever such autonomous adaptations have an impact on the outputted QoS of other coalition members the need of coordination arises: how to ensure that local, individual, adaptation actions of a node can produce a globally acceptable solution for the entire distributed service [7].

Yet, coordinating autonomous dependent adaptations in an open and dynamic system is challenging and may require complex communication and synchronisation strategies. According to [8] it borrows from as diverse areas as computer science, organisation theory, operations research, economics, linguistics, and psychology. This great diversity makes it very difficult to discuss every potential work that has some remote relation to coordination.

Researchers are increasingly investigating decentralised coordination models to establish and maintain system properties [10,11,12,13,14]. A system built using a decentralised coordination model is a self-organising system whose system-wide behaviour is established and maintained solely by the interactions of its nodes that execute using only a partial view of the system [9]. How these nodes were to interact in order to solve the problem (and not what they were actually doing) became the focus of decentralised coordination research. Without a central coordination entity, the collective adaptation behaviour must emerge from local interactions among nodes. This is typically accomplished through the exchange of multiple messages to ensure that all involved nodes make the same decision about whether and how to adapt.

Bridges et al. [15] propose a framework based on Cactus [16] which supports adaptations that span multiple components and multiple nodes in a distributed system. The architecture supports coordinated adaptations across layers using a fuzzy logic based controller module and coordination across hosts using a prototype protocol designed for communication oriented services. However, the authors do not specifically propose a method to obtain a globally coordinated solution.

Ren et al. [17] present a real-time reconfigurable coordination model (RT-RCC) that decomposes dynamic real-time information systems based on the principle of separation of concerns, namely, functional actors which are responsible for accomplishing tasks, and non-functional coordinators which are responsible for coordination among the functional actors. High level language abstractions and a framework for actors and coordinators are provided to facilitate programming with the RT-RCC model.

A similar approach is followed by Kwiat et al. [18] who propose a coordination model for improving software system attack-tolerance and survivability in open hostile environments. The coordination model is based on three active entities: actors, roles,

and coordinators. Actors abstract the system's functionalities, while roles and coordinators statically encapsulate coordination constraints and dynamically propagate those constraints among themselves and onto the actors. Both the coordination constraints and coordination activities are distributed among the coordinators and roles, shielding the system from single points of failure.

However, none of these works proposes support for coordinating inter-dependent autonomous QoS adaptations in cooperative systems, which is the focus of our work. Furthermore, with some decentralised coordination models it becomes difficult to predict the exact behaviour of the system taken as a whole because of the large number of possible non-deterministic ways in which the system can behave [19].

Whenever real-time decision making is in order, a timely answer to events suggests that after some finite and bounded time we would expect the global adaptation process to converge to a consistent solution. Furthermore, optimal decentralised control is known to be computationally intractable [12], although near-optimal systems can be developed for certain classes of applications [20,21,14].

Our goal is then to achieve a fast convergence to a global solution through a regulated decentralised coordination without overflowing nodes with messages. The next section details the proposed one-step decentralised coordination model based on an effective feedback mechanism to reduce the complexity of the needed interactions among nodes until a collective adaptation behaviour is determined.

3.1 A one-step decentralised coordination model

The core idea behind the proposed decentralised coordination model is to support distributed systems composed of autonomous individual nodes working without any central control but still producing the desired function as a whole.

We model a self-managed coalition as a group of nodes that respond to environmental inter-dependent changes according to a distributed coordination protocol defined by the following phases:

1. **Coordination request.** Whenever $Q^i_{val'}$, the needed downgrade or desired upgrade of the currently outputted QoS Q^i_{val} for a work unit $w_i \in S$, has an impact on the currently supplied QoS level of other work units $w_j \in S$ being executed at other coalition members, a coordination request is sent to the affected partners.
2. **Local optimisation.** Affected partners become aware of the new output values $Q^i_{val'}$ of w_i and recompute their local set of SLAs in order to formulate the corresponding feedback on the requested adaptation action. We assume that coalition partners are willing to collaborate in order to achieve a global coalition's consistency, even if this might reduce the utility of their local optimisations. However, a node only agrees with the requested adaptive action if and only if its new local set of SLAs is feasible.
3. **Adaptive action.** If the requested adaptive action is accepted by all the affected nodes in the coalition, the new local set of SLAs is imposed at each of those coalition members. Otherwise, the currently global QoS level of service S remains unchanged.

As such, requests for coordination may dynamically arrive at any time, at any node n_j. We consider the existence of feasible QoS regions [22]. A region of output quality $[q(o)_1, q(o)_2]$ is defined as the QoS level that can be provided by a work unit when provided with sufficient input quality and resources. Within a QoS region, it may be possible to keep the current output quality by compensating for a decrease in input quality by an increase in the amount of used resources or vice versa.

As such, if a node n_j, despite the change in current quality of some or all of its inputs, is able to continue to produce its current QoS level there is no need to further propagate the required coordination request along the dependency graph $\mathcal{G}_{\mathcal{W}}$. Thus, a *cut-vertex* is a key node in our approach.

Consider that a node n_k proposes an upgrade to Q'_{val} for a work unit $w_k \in S$. It may happen that some other nodes, precedent in the path until the next cut-vertex n_c, may be able to upgrade to Q'_{val} and others may not. Whenever the cut-vertex n_c receives the upgrade request and its new set of inputs, if it is unable to upgrade to Q'_{val} then all the effort of previous nodes to upgrade is unnecessary and the global update fails. Otherwise, the upgrade coordination request continues along the graph, until the end-user node n_u is reached.

In the case of a downgrade to Q'_{val} initiated by node n_k, it may happen that some other nodes in the path to the next cut-vertex n_c may be able to continue to output their current QoS level despite the downgraded input and others may not. Again, if the cut-vertex is unable to keep outputting its current QoS level then all the precedent nodes which are compensating their downgraded inputs with an increased resource usage can downgrade to Q'_{val} since their effort is useless.

In these and all other cases, the formulation of the corresponding positive or negative feedback, at the *local optimisation* phase, must depend on the feasibility of the requested coordination action as a function of the quality of the node's new inputs I_{w_i} for the locally executed work unit w_i. Such feasibility is determined by an anytime local QoS optimisation algorithm [23] which aims to minimise the impact of the requested changes on the currently provided QoS level of other services.

Note that the node's local resource usage optimisation associated with the new local set of SLAs can be lower after the coordination request. Recall that we make the assumption that, in cooperative environments, coalition partners are willing to collaborate in order to achieve a global coalition consistency, even if this coordination might reduce the global utility of their local QoS optimisations.

If all the nodes affected by the requested adaptation sent by node n_i agree with its new service solution, the *adaptive action* phase takes place. A "commit" message is sent by node n_i to its direct neighbours in the dependency graph, which then propagate the message to all the involved nodes in the global adaptation process.

Decentralised control is then a self-organising emergent property of the system. The proposed coordination model is based on these two basic modes of interaction: positive and negative feedback. Negative feedback loops occur when a change in one coalition member triggers an opposing response that counteracts that change at other dependent node. On the other hand, positive feedback loops promote global adaptations. The snowballing effect of positive feedback takes an initial change in one node and reinforces that change in the same direction at all the affected partners. By exchanging

feedback on the performed self-adaptations, nodes converge towards a global solution, overcoming the lack of a central coordination and global knowledge.

Note that only one negotiation round is required between any pair of dependent nodes. As such, the uncertain outcome of iterative decentralised control models whose effect may not be observable until some unknowable time in the future is not present in the proposed regulated coordination model.

Also note that the normal operation of nodes continues in parallel with the *change acknowledge* and *local optimisation* phases. Every time a node recomputes its set of local SLAs, promised resources are pre-reserved until the global negotiation's outcome is known (or a timeout expires). As such, the currently provided QoS levels only actually changes at the *adaptive action* phase, as a result of a successful global coordination.

Due to the environment's dynamism, more than one coalition member can start an adaptation process that spans multiple nodes at a given time. Such request can either be a downgrade or an upgrade of its current SLA for a work unit w_i of service S. Even with multiple simultaneous negotiations for the same service S, only one of those will result in a successful adaptation at several nodes since, due to local resource limitations, only the minimum globally requested SLA will be accepted by all the negotiation participants. In order to manage these simultaneous negotiations, every negotiation has an unique identifier, generated by the requesting node.

4 Evaluation

The simulator used in the conducted experiments was custom built in Erlang, a functional programming language designed to be run in a distributed environment populated with resource constrained devices. An application that captures, compresses and transmits frames of video to end users, which may use a diversity of end devices and have different sets of QoS preferences, was used to evaluate the behaviour of the proposed decentralised coordination model, with a special attention being devoted to introduce a high variability in the characteristics of the considered scenarios. The application is composed by a set of source units to collect the data, a compression unit to gather and compress the data sent from multiple sources, a transmission unit to transmit the data over the network, a decompression unit to convert the data into the user's specified format, and an user unit to display the data in the end device.

The number of simultaneous nodes in the system randomly varied from 10 to 100. Each node was running a prototype implementation of the CooperatES framework, with a fixed set of mappings between requested QoS levels and resource requirements. The code bases needed to execute each of the application's units was loaded a priori in all the nodes.

The characteristics of end devices and their more powerful neighbour nodes was randomly generated, creating a distributed heterogeneous environment. This non-equal partition of resources affected the ability of some nodes to singly execute some of the application's units and has driven nodes to a coalition formation for a cooperative service execution.

At randomly selected end devices, new service requests from 5 to 20 simultaneous users were randomly generated, dynamically generating different amounts of load and resource availability during the previously accepted services' execution.

Each service request was formulated as a set of random QoS levels, expressing the spectrum of the user's acceptable QoS levels in a qualitative way, ranging from a randomly generated desired QoS level to a randomly generated maximum tolerable service degradation. The relative decreasing order of importance imposed in dimensions, attributes and values was also randomly generated.

Based on each user's service request, coalitions of 4 to 20 nodes were formed using the anytime coalition formation and service proposal formulation algorithms proposed in [24,25]. Each node was connected at least to another node in the coalition. The maximum degree of each node, that is, the number of connections to a node was set to 3. After the coalition was formed, a randomly percentage of the connections among its members was selected as a QoS dependency among those work units.

To cope with such a dynamic environment, nodes were requested to adjust their local set of SLAs, either by lowering the currently provided QoS level of some services due to resource limitations or by (re)upgrading them when the needed resources become available [25].

The behaviour of the proposed decentralised one-step coordination model in highly dynamic scenarios was compared to a classic centralised optimal coordination model. With a centralised coordination model, all changes in the output quality of a work unit $w_{ij} \in S_i$ have to be communicated to a single entity with service-wide knowledge. Then, this central coordinator has to determine the impact of those changes in the overall coalition's QoS level and request the adaptation of the involved nodes. To evaluate the success or failure of such dependent adaptation, an adaptation request must be sequentially made along the dependency graph either until a common global service solution is found or one of the coalition member is unable to supply the new desired QoS values.

The conducted evaluation started by comparing the total number of messages that had to be exchanged among nodes when using both approaches to globally coordinate dependent autonomous self-adaptations. The average results of all simulation runs for different coalition sizes are plotted in Figure 1.

As expected, both coordination approaches require more messages to be exchanged among nodes as the complexity of the service's topology increases. Nevertheless, the proposed decentralised coordination model requires around 80% of the needed number of messages required by the centralised model until all the affected coalition members become aware of the coordination request's result.

Less messages should result in a faster convergence to a global common solution. To verify the veracity of such assumption a second study measured the needed average time from the moment a node issued a coordination request until the outcome of the global adaptation process was determined. The deadline used for the anytime local QoS adaptation at each node was set to one second. At the end of the algorithm's execution, the feasibility or unfeasibility of the received coordination request was determined by the node. The obtained results are plotted in Figure 2.

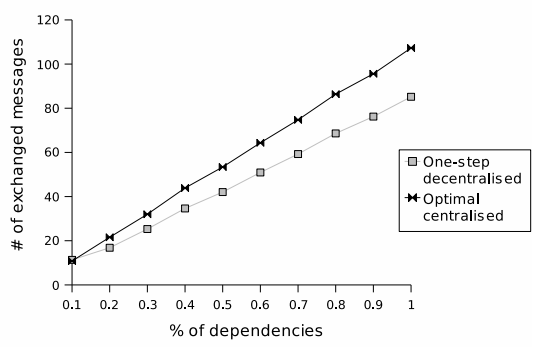


Fig. 1. Average number of exchanged messages

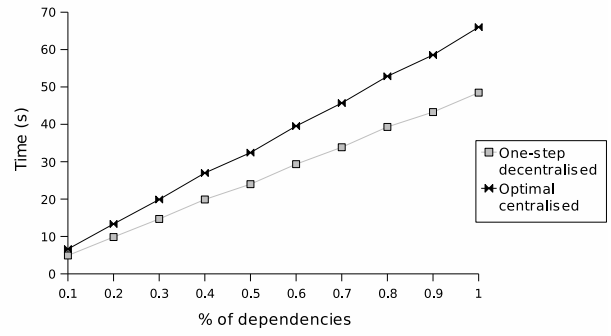


Fig. 2. Needed time until the global adaptation result is determined

Clearly, the proposed decentralised coordination model is faster to determine the overall coordination result in all the evaluated services' topologies, needing approximately 75% of the time spent by the centralised optimal model.

Even if the proposed one-step decentralised model requires less messages and is faster than a centralised optimal model to determine a global solution it is still important to evaluate impact of an one-step coordination model on the achieved service solution's quality. Recall that, when adopting the proposed one-step coordination algorithm, if some other dependent node in the coalition is unable to supply the new requested values no other alternative solution is tried and the global adaptation process fails. On the other hand, with the centralised optimal coordination model, a node is able to reply with a service counter-proposal whenever it is unable to coordinate with the currently requested values. As such, it is possible that after some iterations, the node's best possible service solution can be accepted by all the dependent coalition partners as part of

a global SLA. Note that such intermediate service solution would not be achieved with the proposed one-step coordination model.

The reward of each determined SLA after a successful global coordination process was evaluated by computing, for each service's QoS dimension, a weighted sum of the differences between the user's preferred quality values and the proposed values [5,24]. The results were plotted, in Figure 3, by averaging the results over several independent runs of the simulation, divided in two categories: (i) when the average amount of available resources per node is greater than the average amount of resources demanded by the services being executed; and (ii) when the average amount of resources per node is smaller than the average amount of demanded resources.

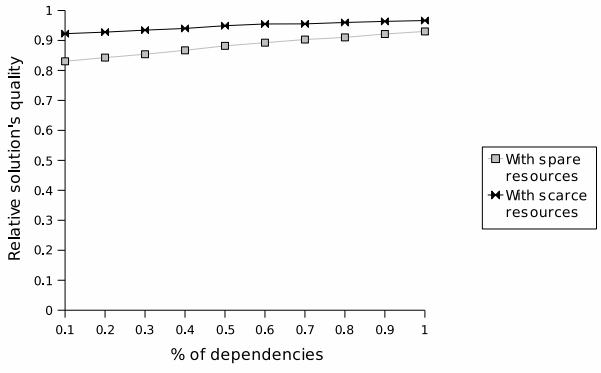


Fig. 3. Relative solution's utility as a function of available resources

As the coalition's topology complexity increases it is clearly noticeable, in both scenarios, that a near-optimal service solution's quality is achieved when using the one-step coordination model, despite its simpler approach and faster convergence to a common solution. The achieved results can be explained by the fact that as the coalition's topology complexity increases it also increases the probability of one of the involved nodes in the global adaptation process to be unable to use more than its current level of reserved resources for a work unit $w_i \in S$. As the achieved results clearly demonstrate, such probability is even greater when the resources are scarce.

5 Conclusion

In heterogeneous open distributed environments, computing devices can range from small, resource limited, mobile devices to stationary powerful servers. As such, for some of those nodes, there may be a constraint on the type and size of applications that can be executed with specific user's acceptable quality of service (QoS) and drive nodes to a coalition formation for a cooperative service execution.

This paper addressed the problem of coordinating autonomous dependent adaptations of resource-bounded nodes in dynamic open cooperative real-time environments. The proposed one-step decentralised coordination model imposes restrictions on the way members of dynamically created coalitions can self-adapt in response to changes in the environment.

The proposed coordination model is based on an effective feedback mechanism that reduces the complexity of the needed interactions among nodes. Feedback is formulated as a result of a QoS adaptation process that evaluates the feasibility of the new requested service solution. As the achieved results clearly demonstrate, the proposed coordination model has a reduced overhead and enables a faster convergence to a near-optimal global service solution.

Acknowledgements

This work was supported by FCT through the CISTER Research Unit - FCT UI 608 and the research project CooperatES - PTDC/EIA/71624/2006.

References

1. Allen, G., Dramlitsch, T., Foster, I., Karonis, N.T., Ripeanu, M., Seidel, E., Toonen, B.: Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In: Proceedings of the 2001 ACM/IEEE conference on Supercomputing. (November 2001) 52–52
2. Ensink, B., Adve, V.: Coordinating adaptations in distributed systems. In: Proceedings of the 24th International Conference on Distributed Computing Systems, Tokyo, Japan (March 2004) 446–455
3. Montresor, A., Meling, H., Babaoglu, Ö.: Toward self-organizing, self-repairing and resilient distributed systems. In: Future Directions in Distributed Computing. (2003) 119–126
4. Rajkumar, R., Lee, C., Lehoczky, J., Siewiorek, D.: A resource allocation model for qos management. In: Proceedings of the 18th IEEE Real-Time Systems Symposium, IEEE Computer Society (1997) 298
5. Nogueira, L., Pinho, L.M.: Dynamic qos-aware coalition formation. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Denver, Colorado (April 2005) 135
6. Burgess, M.: On the theory of system administration. Science of Computer Programming **49** (2003) 1
7. Gelernter, D., Carriero, N.: Coordination languages and their significance. Communications of the ACM **35**(2) (1992) 96–107
8. Malone, T.W., Crowston, K.: The interdisciplinary study of coordination. ACM Computing Surveys **26**(1) (1994) 87–119
9. Goldin, D., Keil, D.: Toward domain-independent formalization of indirect interaction. In: Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Washington, DC, USA, IEEE Computer Society (2004) 393–394
10. Dorigo, M., Caro, G.D.: The ant colony optimization meta-heuristic. New ideas in optimization (1999) 11–32

11. Graupner, S., Andrzejak, A., Kotov, V., Trinks, H.: Adaptive control overlay for service management. In: First Workshop on the Design of Self-Managing Systems, San Francisco, USA (June 2003)
12. De Wolf, T., Holvoet, T.: Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control. Proceedings of the IEEE International Conference on Industrial Informatics (August 2003) 470–479
13. Boutilier, C., Das, R., Kephart, J.O., Tesauro, G., Walsh, W.E.: Cooperative negotiation in autonomic systems using incremental utility elicitation. In: In Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence, Acapulco,Mexico (August 2003) 89–97
14. Dowling, J., Haridi, S.: in Reinforcement Learning: Theory and Applications. In: Decentralized Reinforcement Learning for the Online Optimization of Distributed Systems. I-Tech Education and Publishing, Vienna, Austria (2008) 142–167
15. Bridges, P., Chen, W.K., Hiltunen, M., Schlichting, R.: Supporting coordinated adaptation in networked systems. In: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, Oberbayern, Germany (May 2001) 162
16. The University of Arizona, C.S.D.: The cactus project. Available at `http://www.cs.arizona.edu/projects/cactus/`
17. Ren, S., Shen, L., Tsai, J.: Reconfigurable coordination model for dynamic autonomous real-time systems. In: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, Taichung, Tawain (June 2006) 60–67
18. Kwiat, K., Ren, S.: A coordination model for improving software system attack-tolerance and survivability in open hostile environments. In: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, Taichung, Tawain (June 2006) 394–402
19. Serugendo, G.D.M.: Handbook of Research on Nature Inspired Computing for Economy and Management. In: Autonomous Systems with Emergent Behaviour. Idea Group, Inc., Hershey-PA, USA (September 2006) 429–443
20. Jelasity, M., Montresor, A., Babaoglu, O.: A modular paradigm for building self-organizing peer-to-peer applications. In: In Engineering Self-Organising Systems, G. Di Marzo Serugendo, Springer (2004) 265–282
21. Dusparic, I., Cahill, V.: Research issues in multiple policy optimization using collaborative reinforcement learning. In: Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems, Washington, DC, USA, IEEE Computer Society (2007) 18
22. Shankar, M., de Miguel, M., Liu, J.W.S.: An end-to-end qos management architecture. In: Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium, Washington, DC, USA, IEEE Computer Society (1999) 176–191
23. Nogueira, L., Pinho, L.M.: Time-bounded distributed qos-aware service configuration in heterogeneous cooperative environments. Journal of Parallel and Distributed Computing **69**(6) (June 2009) 491–507
24. Nogueira, L., Pinho, L.M.: Iterative refinement approach for qos-aware service configuration. IFIP From Model-Driven Design to Resource Management for Distributed Embedded Systems **225** (2006) 155–164
25. Nogueira, L., Pinho, L.M.: Dynamic qos adaptation of inter-dependent task sets in cooperative embedded systems. In: Proceedings of the 2nd ACM International Conference on Autonomic Computing and Communication Systems, Turin,Italy (September 2008) 97

RTEMS Improvement – Space Qualification of RTEMS Executive

Helder Silva, José Sousa, Daniel Freitas, Sergio Faustino,
Alexandre Constantino and Manuel Coutinho,

EDISOFT, Empresa de Serviços e Desenvolvimento de Software, S.A.,
Rua Quinta dos Medronheiros - Lazarim,
Apartado 2826-801 Caparica, Portugal
{Helder.Silva, José.Sousa, Daniel.Freitas, Sergio.Faustino,
Alexandre.Constantino, Manuel.Coutinho}@Edisoft.pt

Abstract. RTEMS (Real-Time Executive for Multiprocessor Systems) has been selected as a possible ESA (European Space Agency) building block for space missions. As a first step to achieve the category of a building block, RTEMS must attain TRL (Technological Readiness Level) 6, which is equivalent to a product release in software engineering. TRL6 is achieved after the successful evaluation of the RTEMS technology, in this case, after the conclusion of RTEMS Improvement project. The full qualification of the technology will be achieved after the successful run of the operating system qualification process, with the space mission hardware and the application running on top of RTEMS, like control systems, telecommand or telemetry applications.

Keywords: RTEMS, Space Qualification, Galileo Software Standards, Real-Time and Embedded Systems.

1 Introduction

This paper presents the current outputs of an industrial activity held by EDISOFT to facilitate the qualification of RTEMS (Real-Time Executive for Multiprocessing Systems) for space applications and space missions. The activity has been performed with and for ESA (European Space Agency) following the GSWS (Galileo Software Standards) for a DAL (Development Assurance Level) allocation B (software whose anomalous behaviour would cause or contribute to a failure resulting in a critical event).

This paper summarizes the presentations made in the last 3 years in DAta Systems in Aerospace conference in 2007, [1], 2008, [2] and 2009, [3], which describe the activities performed in ESA contracts [6] and [7] of the RTEMS CENTRE project (Support and Maintenance CENTRE to RTEMS Operating System), final presentation made in ESTEC (European Space Research and Technology Centre) in December 2008 [4] and RTEMS Improvement project, an on-going ESA project.

RTEMS CENTRE objectives were the design, development, maintenance and integration of tools to augment and sustain RTEMS operating system and the creation

and maintenance of technical competences and support site to RTEMS operating system in Europe. The general idea is to minimize the cost of airborne and space applications incorporation/integration.

RTEMS Improvement primary objectives are to improve RTEMS product and its documentation in order to facilitate the qualification of RTEMS for future space missions, taking into account the specific operational requirements and target environment and to provide Memory Manager for LEON hardware Memory Management Unit (MMU).

The first section makes the introduction of the paper, the rationale behind a qualification process and the description of each of the sections. Section 2 makes a brief presentation of the RTEMS executive, section 3 provides an overview of the qualification process and section 5 concludes the paper and highlights some of the future work of EDISOFT in RTEMS.

2 RTEMS Overview

Real Time Executive for Multiprocessor Systems (RTEMS) is an open source Real Time Operating System (RTOS) designed for deeply embedded systems that aim to be competitive with closed source and commercial products [9]. Currently it is maintained by the On-Line Research Corporation (OAR) albeit many of the features and platform support for it have been developed by RTEMS users. The first version, of what is today RTEMS, was released in 1988.

RTEMS supports multiple processors, including SPARC (ERC32 and LEON family), i386, Power PC and others, complies with several standards (POSIX, RTEID/ORKID, TCP/IP, µITRON, ANSI C/C++, partially with Ada95), supports basic kernel features, provides networking (FreeBSD TCP/IP stack, UDP, TCP, etc) and filesystem functionalities (IMFS, FAT 32/16/12, etc), and includes debugging features (over Ethernet and serial port). RTEMS was designed to support applications with the most inflexible time frames requirements, making it possible for the user to develop hard real time systems [10].

The RTEMS kernel supports several features. The most important are multi-tasking, networking and support of file systems.

Multi-tasking allows the software to be more responsive and modular, but brings a lot of issues that must be solved by the kernel. RTEMS makes possible to use multi-tasking, since it implements three scheduling features (Event driven, Priority driven and Rate monotonic). The kernel also allows the selection of the modules that are loaded, avoiding unnecessary delays and memory usage (footprint).

Networking features are also quite valuable, since RTEMS implements several useful networking protocols. This allows loading a kernel or executive from a network, using BOOTP (Bootstrap Protocol). This avoids the need to have a complete operating system running in the target just to load a new executive since it's loaded from the network. Other protocols are useful for the application itself, like TCP, UDP, ICMP, RARP and DHCP. There are also some servers implemented as FTP server, HTTP server that allows file transfer in an easy manner. The PPP server allows the connection via dial-up modems and Telnet to control and configure the target system.

RTEMS filesystem provides features like UNIX, e.g., mountable systems, hierarchical system directory structure, POSIX compliant set of routines for the manipulation of files and directories.

RTEMS remote debugging is very important, since debugging on site is difficult due to board constraints. For development itself, RTEMS can build applications in C and C++, following the ISO standards, and ADA95 (limited). POSIX and µITRON standards are implemented in the system API's.

RTEMS can be characterized into three distinct levels [10], the operational, organizational and conceptual levels.

The operational level defines the relationship between RTEMS and the user application. The RTEMS installation process ends with two major libraries (librtemsbsp.a and librtemscpu.a) that are linked with the user application.

The organization level is the approach taken by RTEMS developers to organize and structure the source code of the operating system. Fig. 1 presents the top level RTEMS directory structure.

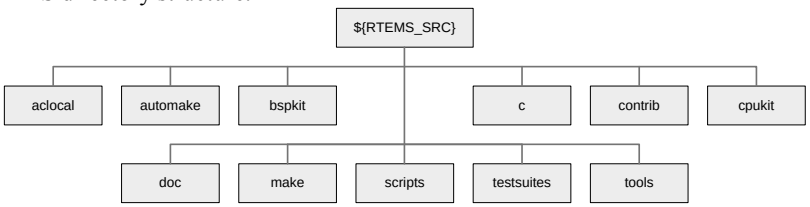


Fig. 1. RTEMS Directory Structure

The conceptual level is divided in three different layers, the hardware support, the kernel and the application interface. Applications can be developed in C, C++ and Ada95 (support is limited for Ada95) using different APIs such as Ada, POSIX, µITRON and RTEMS' own API set (based on the RTEID/ORKID standard). All APIs use RTEMS supercore, except for the Ada API which uses directly the RTEMS' API as an abstraction layer. The hardware support layer encompasses the processor and board dependent files as well as a common hardware library. The kernel layer is the heart of RTEMS and encompasses the super core, the super API and several portable support libraries. Fig. 2 presents a schematic of the RTEMS conceptual level architecture.

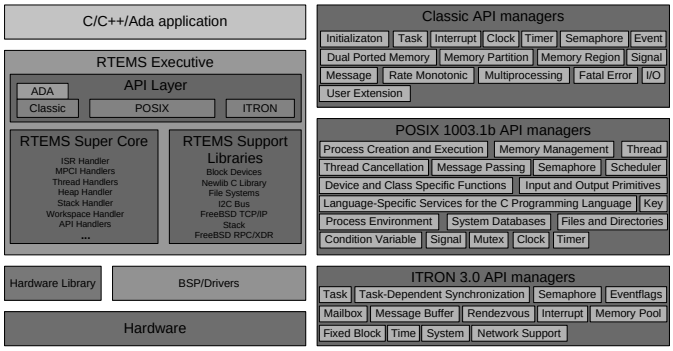


Fig. 2. RTEMS Architecture

3 Qualification Process

The qualification process aims to increase the quality of any product and its associated documentation. This prevents the occurrence of future problems with the software. The qualification deliverables were produced for RTEMS version 4.8.0 and for ERC32 (SPARC V7 hardware architecture), LEON2 and LEON3 (SPARC V8 architecture), the hardware boards used and produced by ESA for the space missions. The qualification process can be made using some of the software engineering standards like DO-178B, ECSS-E40, ECSS-Q80 and GSWS. Since it was predicted the usage of RTEMS in Galileo program and since the GSWS is the most complete standard known in terms of qualification, ESA and EDISOFT decided the usage of GSWS.

3.1 Galileo Software Standards

The Galileo Software Standard (GSWS) [5] sets requirements to be followed for the management, evaluation, procurement, development, production, verification, operation and maintenance of all software products.

It defines procedures to be followed for software engineering, software product assurance and software configuration management. Several reference life cycles for the SW Components are defined. Table 1 presents the generic software waterfall life cycle used for RTEMS Improvement.

Table 1. Generic software waterfall life cycle phases and reviews

Phase	SW Review
SW Planning Phase	Software Requirements Review
SW Specification Phase	Preliminary Design review
SW Design Phase	Detailed Design Review
SW Implementation Phase	Test Readiness Review
SW Integration and TS-Validation Phase	Critical Design Review
SW RB-Validation Phase	Qualification Review
SW Acceptance Phase	Acceptance Review
SW Maintenance Phase	--
SW Operation Phase	--

The software is then classified with a DAL (Development Assurance Level) that is dependent of the criticality usage of the software. The levels go from Level A, the highest level of criticality, to Level E, the lowest level of criticality. RTEMS was classified as DAL B, since its anomalous behaviour can cause a failure resulting in a critical event. Based in the DAL selection of the software, the development process, the deliverables and tests to be performed are fully defined, meaning that each Level has its own development life-cycle, document deliverables, phases and document versions. Table 2 makes an overview of GSWS software life cycles, type of software and DAL.

Table 2. GSWS Life Cycles vs. DAL and Types of Software

Type Of Software	DAL A	DAL B	DAL C	DAL D	DAL E
Generic	Waterfall	Waterfall	Waterfall	Waterfall Incremental	Waterfall Incremental
Databases	Waterfall	Waterfall	Waterfall	Waterfall	Waterfall
MMI	Evolutionary	Evolutionary	Evolutionary	Evolutionary	Evolutionary
Test Software	--	--	--	--	Waterfall Incremental
Simulators	Waterfall	Waterfall	Waterfall	Waterfall	Waterfall
Algorithm Prototypes	--	--	--	--	Evolutionary

The following figures (Fig. 3, Fig. 4 and Fig. 5) present a sample of the different life cycles used in GSWS. In the waterfall life cycle, the one used in RTEMS Improvement, the phases are produced continuously and each phase is marked by a review meeting.

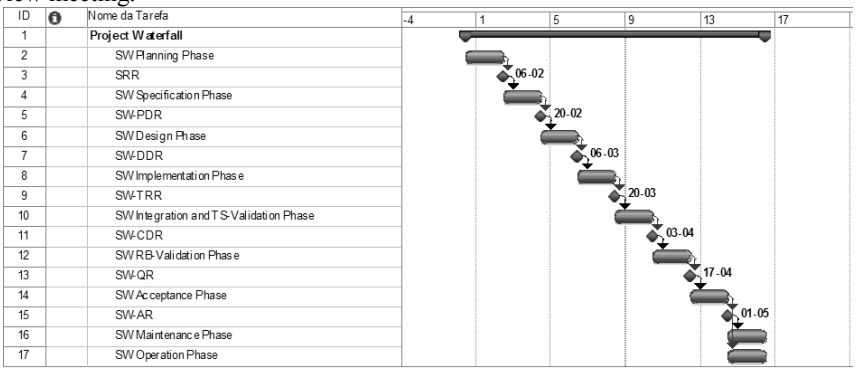


Fig. 3. Waterfall Project

In the evolutionary life-cycle, the software is built in different builds of design, implementation, integration and validation, ending by a QR and followed by the acceptance phase.

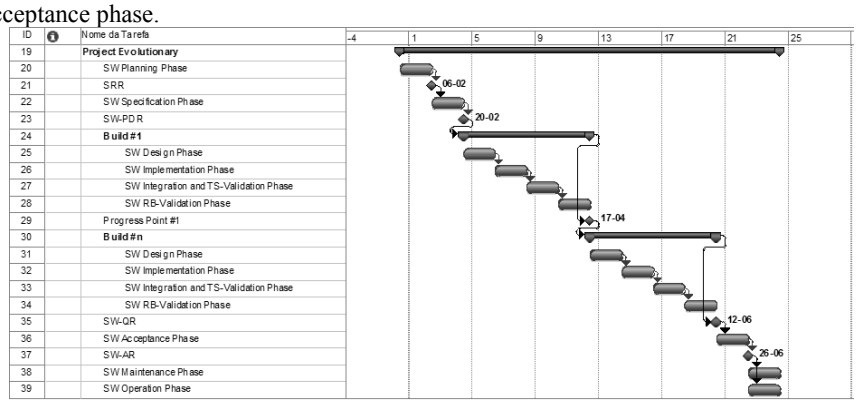


Fig. 4. Evolutionary Project

The incremental life-cycle is also built in different builds of just design, implementation, integration and a pre-validation. A formal validation of all the builds is necessary.

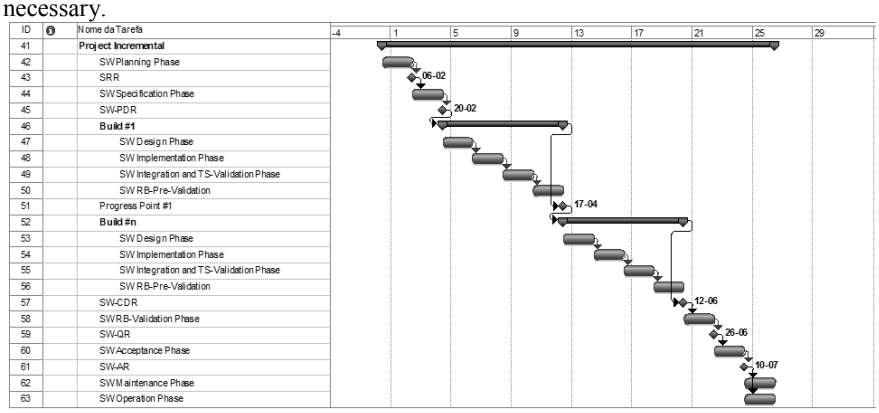


Fig. 5. Incremental Project

3.2 RTEMS Qualification Overview

The main objective is to provide a *qualifiable* version of RTEMS. The RTEMS version 4.8.0 was selected as baseline for the project because it was the latest version of RTEMS and because it added and fixed important features, like support for time granularity in nanoseconds, fixed problems on thread's priority, reduced the footprint for a more compact executable and presented a set of new drivers for LEON2 and LEON3 processors.

The user is capable of downloading RTEMS and with a configuration tool, is capable of filtering relevant RTEMS features and removing dead code (applying patch to the original RTEMS). At the same time, the same tool builds a tailored test suite that is merged and executed with the tailored version of RTEMS to produce the final system (operating system sub-set and test case battery) that facilitates the qualification of the operating system software. Fig. 6 presents a general overview of final product of the RTEMS Improvement.

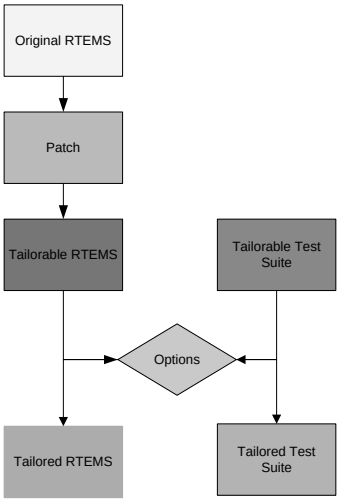


Fig. 6. RTEMS Tailoring Road Map

Documentation of RTEMS was revised and used as input to create the RTEMS requirements [16], to reverse engineer RTEMS and to build its architecture [15]. The design of RTEMS was addressed and constitutes the base for the development and design of the test battery. Table 3 presents a list of the RTEMS documentation produced so far by the RTEMS Improvement.

Table 3. List of RTEMS produced Documentation

RTEMS Improvement Data Pack
RTEMS managers candidate evaluation report
RTEMS Improvement Requirement Document
RTEMS Improvement User Manual and Design Notes
RTEMS Improvement Verification Report
Software Budget Report
Product Software Justification File
RTEMS Improvement Design Document
RTEMS Improvement Configuration File
RTEMS Improvement Integration Test Plan
RTEMS Improvement Unit Test Plan
RTEMS Improvement Validation Testing Specification – Technical Specification
RTEMS Tailoring Plan
RTEMS Improvement Generic Test Report
RTEMS Test Suite
RTEMS Improvement Acceptance Test Plan
RTEMS Improvement Maintenance Plan
RTEMS Improvement Installation Report
RTEMS Improvement Acceptance Data Package
RTEMS Tailored
Software Development Plan
Review Plan
Final Report

RTEMS Improvement Product Assurance Plan
RTEMS Improvement Product Assurance Report
RTEMS Improvement Configuration Management Plan
RTEMS Improvement SOC with GSWS
RTEMS Improvement Preliminary Software Criticality Analysis Report

3.3 RTEMS Candidate Managers

Surveys [11] were performed near European Space users (SAAB, OHB and ESA) and along with the EDISOFT assessment, candidate RTEMS managers were selected. Table 4 provides the list of the managers currently being used in some space projects by SAAB and OHB. This information provides RTEMS Improvement guides for the facilitation of qualification.

Table 4. Space Users Survey Results

RTEMS Managers	SAAB Survey	OHB Survey	ESA SoW
Initialization Manager	Yes	Yes	Yes
Task Manager	Yes	Yes	Yes
Interrupt Manager	Yes	Yes	Yes
Clock Manager	Yes	Yes	Yes
Timer Manager	Yes	Yes	Yes
Semaphore Manager	Yes	Yes	Yes
Message Manager	Yes	Maybe	Yes
Event Manager	Yes	Maybe	Yes
Signal Manager	No	Maybe	Yes
Partition Manager	Yes	Maybe	No
Region Manager	No	Maybe	No
Dual-Ported Memory Manager	No	No	No
I/O Manager	No	Yes	Yes
Fatal Error Manager	Yes	Yes	Yes
Rate Monotonic Manager	Yes	Yes	Yes
Barrier Manager	No	Maybe	No
User Extensions Manager	No	No	Yes
Multiprocessing Manager	No	No	Yes
Stack Bounds Checker	No	No	No
CPU Usage Statistics	No	No	No

Based in the survey conducted and a deep analysis of the RTEMS Classic API Managers and its dependencies, it was possible to select the candidate managers and primitives to be included in the work. Table 5 displays the results.

Table 5. Selected RTEMS Managers

RTEMS Manager	RTEMS Primitive	RTEMS Manager	RTEMS Primitive
Initialization	All directives	Clock	All directives
Task	rtems_task_create	Timer	All directives
	rtems_task_ident	Semaphore	All directives
	rtems_task_start	Message Queue	All directives
	rtems_task_restart	Event	All directives
	rtems_task_delete	I/O	rtems_io_initialize
	rtems_task_suspend		rtems_io_open
	rtems_task_resume		rtems_io_close
	rtems_task_is_suspended		rtems_io_read
	rtems_task_set_priority		rtems_io_write
	rtems_task_mode		rtems_io_control
	rtems_task_get_note	Fatal Error	All directives
	rtems_task_set_note	Rate Monotonic	rtems_rate_monotonic_creat
	rtems_task_wake_after		rtems_rate_monotonic_ident
	rtems_task_wake_when		rtems_rate_monotonic_cance
	rtems_task_variable_add		rtems_rate_monotonic_delet
	rtems_task_variable_get		rtems_rate_monotonic_perio
	rtems_task_variable_delete		rtems_rate_monotonic_get_status
Interrupt	All directives	User Extensions	All directives

3.4 RTEMS Engineering and Testing

The original version of RTEMS was truncated and several files were removed because of two main reasons, they were considered unnecessary and they were dead code. As part of the main goal, one of the project outputs is to provide means to achieve a RTEMS tailored version starting from the RTEMS original version. The version shall run in LEON2, LEON3 and ERC32 platforms (Fig. 7).

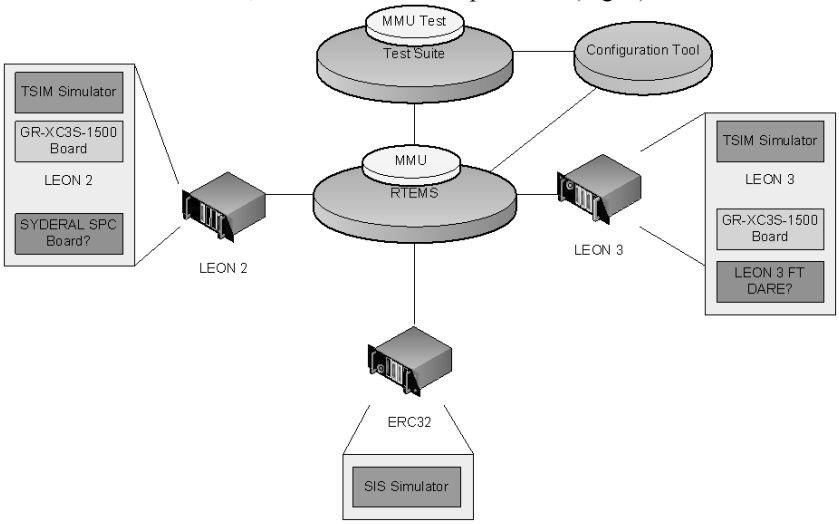


Fig. 7. RTEMS Engineering and Testing Overview

The tailored RTEMS version consists of patches and scripts that, if applied to the original RTEMS source code, will remove the unnecessary managers, files, dead code and bugs. It also adds new files and code, making all necessary code adjustments to produce the RTEMS tailored version (qualifiable). This version intends to achieve the Galileo Software Standards Development Assurance Level (DAL) B requirements. According to the standard, the structural coverage for a DAL-B qualification shall achieve 100% statement and decision coverage for the source code. Based in these requirements, the source code cannot contain dead or unused code. The current coverage of the tests are 86,1% of statement coverage and 3.228 LOC (lines of code) (API, supercore, RTEMS API and Super API) [12]. The original version of RTEMS, counting with comments and headers is around 450.000 LOCs.

In the Statement coverage testing, the code is executed in such a manner that every statement in the code is executed at least once. Branch or Decision Coverage testing helps to validate all branches in the code and also validates that no branching leads to abnormal software behaviour.

At a first phase, the code removal was a very sensible operation, since it included the removal of unselected RTEMS Managers and code shared between RTEMS Managers.

In the current phase of the project the development team is producing unit and integration tests to validate the correctness of RTEMS behaviour.

3.5 RTEMS Budget

Software budget analysis [13] was performed to RTEMS. A comparison between the original RTEMS version and RTEMS tailored was made for all hardware architectures (ERC32, LEON2 and LEON3) of interest. The measurements taken were related with interrupt latency timing, interrupt exit timing, context switch timing, maximum CPU usage, RTEMS API directives timing and memory footprint. The measurements were based in ESA requirements and the intent was to make a comparison between the original RTEMS and the RTEMS Tailored for the most useful characteristics of a real-time operating system. Table 6 presents a sample of context switch analysis budget.

Table 6. Timing Analysis for Context Switch

	Maximum Time (microseconds)					
Section	TargetSimERC32		TargetSimLeon2		TargetSimLeon3	
RTEMS Version	Original	Tailored	Original	Tailored	Original	Tailored
Context Switch without FPU	48	48	19	19	19	19
Context Switch with FPU	68	68	20	20	20	20

3.6 RTEMS Criticality Analysis

SW-FMECA (Software Failure Modes Effects and Criticality Analysis) [14] analysis is a bottom-up technique that identifies modes, causes, effects and criticalities of failures on end item (software) performance and their external interfaces. The SW-FMECA main objective is to perform the DAL assessment of each RTEMS Improvement Component/Sub-Component. The impact of a failure can be characterized by a severity classification. Each failure mode was analysed to estimate the severity level.

The SW-FMECA analysis was performed and it was possible to find recommendations to RTEMS Operating System. The following bullets present the major recommendations found:

- When a "Fatal Error" (in any state) occurs, the RTEMS Operating System, before the User Application starts the system, shall switch to a "SAFER" state (APP_SAFE_STATE). The transition from the "APP_SAFE_STATE" to the "BEFORE_INIT" state shall be done through the Operating System re-initialization (performed by the User Application).
- An Error/Event Handler (Log Manager) to receive all errors from the RTEMS Operating System and HW devices shall be foreseen in the next version of RTEMS Operating System, to improve the management of all errors. The User Application shall access to this "Log Manager" to have the RTEMS Operating System and HW devices errors.
- The Rate Monotonic Manager shall define the deadline for each thread on the System, in order to avoid that:
  - One thread blocks the execution of other threads (low priority threads could not be executed and miss their deadlines).
  - The execution of one faulty thread (running in loop not programmed) uses all the resources of the system.
  - The system enters in degraded mode.
- The use of Dynamic Memory by the "Heap Handler" during the Initialization of the RTEMS components like, semaphores, Threads, Message Queues, timers shall be avoided (Rule 20.4 of [17]).
- RTEMS Operating System shall provide the capability to perform PBIT (Power On Built-In-Tests) when the system is initiated, in the HW devices supported in the scope of the project (Clock device and Processor).

4 Conclusions

This paper provided a brief view of the RTEMS Improvement's project activities. The activities were centred in the acquisition of know-how and recently the facilitation of RTEMS qualification. The qualification process is about to be concluded and a new version of RTEMS will be produced. This new version is based in 4.8.0 of RTEMS. In a recent future EDISOFT will develop the Memory Manager for the RTEMS OS for the LEON architecture Memory Management Unit (MMU). The outputs of the work can be accessed through RTEMS Support Platform [8] and the *qualified* version and tools are distributed as open source free package.

References

1. Constantino, A., Silva, H., Mota, M., Zulianello, M.: RTEMS CENTRE – Support and Maintenance CENTRE to RTEMS Operating System, DAta Systems in Aerospace (2007)
2. Silva, H., Constantino, A., Coutinho, M., Freitas, D., Faustino, S., Mota, M., Colaço, P., Zulianello, M.: RTEMS CENTRE – Support and Maintenance CENTRE to RTEMS Operating System, DAta Systems in Aerospace (2008)
3. Silva, H., Constantino, A., Coutinho, M., Freitas, D., Faustino, S., Mota, M., Colaço, P., Sousa, J., Dias, L., Damjanovic, B., Zulianello, M., Rufino, J.: RTEMS CENTRE – Support and Maintenance CENTRE to RTEMS Operating System, DAta Systems in Aerospace (2009)
4. Silva, H., RTEMS CENTRE Final presentation and Final Report, ESTEC (European Space Research and Technology Centre), Noordwijk - Netherlands (2008)
5. GSWS study team: Galileo Software Standards, GAL-SPE-GLI-SYST-A/0092 (2004)
6. EDISOFT: ESA/ESTEC Contract number 20049/05/NL/JD/jk
7. EDISOFT: ESA/ESTEC Contract number 21141/07/NL/JD
8. RTEMS CENTRE website: http://rtemscentre.edisoft.pt
9. RTEMS website: http://www.rtems.com
10. Constantino, A., Freitas, D., Mota, M., Silva, H.: RTEMS CENTRE Software System Specification, RTEMS CENTRE project (2008)
11. Coutinho, M.: RTEMS Managers Candidate Evaluation Report, RTEMS Improvement project (2009)
12. Coutinho, M.: RTEMS Improvement Generic Test Report, RTEMS Improvement project (2009)
13. Freitas, D.: RTEMS Improvement Software Budget Report, RTEMS Improvement project (2009)
14. Dias, L.: RTEMS Improvement Preliminary Software Criticality Analysis Report, RTEMS Improvement project (2009)
15. Colaço, P., Coutinho, M.: RTEMS Improvement Software Design Document, RTEMS Improvement project (2009)
16. Coutinho, M.: RTEMS Improvement Software Requirements Document, RTEMS Improvement project (2009)
17. MIRA Limited: MISRA-C: 2004 Guidelines for the use of C language in critical systems.

Sessão 1C: Segurança de Sistemas e Redes de Computadores

Building an Automaton Towards Protocol Reverse Engineering*

João Antunes and Nuno Neves
{jantunes,nuno}@di.fc.ul.pt

University of Lisboa,
Faculty of Sciences, LASIGE—Portugal

Abstract. The communication between computer systems is dictated by network protocols, which determine how the network components interact with each other. Knowing the specification of a network protocol can greatly improve the security and dependability of both the design of the protocol and the applications implementing it. The specification can be used, for example, to verify if the application's implementation is correct and in accordance, or even to aid in the creation of specific firewall rules or IDS filters to block messages that do not comply with the defined standard.
However, the protocol specification is not always available, which makes assessing the correctness and security of such protocols difficult. Protocol reverse engineering has been used to overcome this problem, by deducing the specification of closed protocols from their utilization alone and without any assumption about their structure or operation. In this paper, we present two different approaches, based on sequence alignment techniques, to build an automaton of a network protocol from network traces.

Resumo. A comunicação entre sistemas computacionais é ditada pelos protocolos de rede que determinam a forma como os componentes de rede interagem entre si. Conhecer a especificação do protocolo de rede pode melhorar profundamente a segurança e confiabilidade, tanto no desenho do protocolo, como nas aplicações que o concretizam. A especificação pode ser usada, por exemplo, para verificar se a concretização da aplicação está correcta e em conformidade, ou ainda para ajudar na criação de regras específicas de firewall ou filtros de IDS para bloquear mensagens que não estejam em conformidade com o padrão definido.
No entanto, a especificação do protocolo nem sempre está disponível, o que dificulta a avaliação da correcção e segurança destes protocolos. Para resolver este problema, foi proposto usar engenharia de reversão para deduzir a especificação de protocolos fechados a partir apenas da sua utilização e sem qualquer suposição sobre a sua estrutura ou funcionamento. Neste trabalho, apresentamos duas abordagens diferentes, com base nas técnicas de alinhamento de sequências, para construir um autómato de um protocolo através do tráfego de rede.

* This work was partially supported by FCT through the Multiannual Funding and the CMU-Portugal Programs.

1 Introduction

Network protocols are essential in today's interconnected world. The communication between computer systems is dictated by a set of rules that regulates the syntax, semantics, and synchronization of the exchanged messages. Since it determines how the network components interact with each other, they are of crucial importance in security-related contexts. Most exploits, for instance, take advantage of vulnerabilities present in network protocols, or at least use them as a vehicle to explore vulnerabilities present in interconnected applications, such as servers or critical infrastructures. Knowing the specification of a network protocol can improve the security and dependability of both the design of the protocol and the applications that implement it. For example, one can verify if a network server correctly complies to the specification of the protocol by creating test cases covering the protocol space and then comparing its responses (and internal state) with the expected ones. The protocol specification can even be used to create firewall rules, denying messages not fully complying with the standard, or to provide intrusion detection systems (IDS) with the capability of performing deep packet inspection.

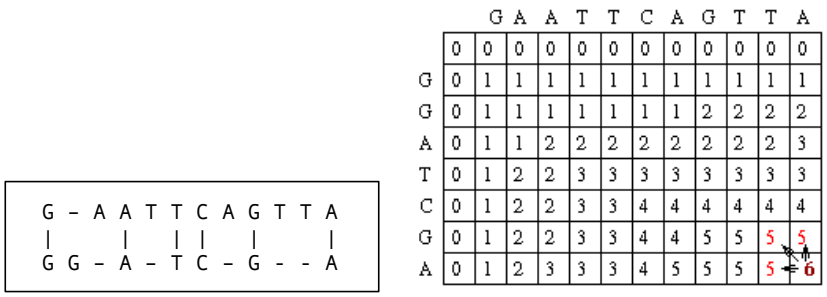
However, some Commercial Off-The-Shelf (COTS) components rely on closed protocols for their execution, sometimes to avoid the integration with other third-party components, other times for security reasons. Nonetheless, security through obscurity is not a safe principle, as a sufficiently motivated attacker will eventually discover the hidden flaws.

Protocol reverse engineering has been used to address this problem, by deducing the specification of closed protocols from their utilization alone and without any assumption about their structure or operation. In this paper, we present two different approaches to build an automaton of a network protocol from network traces. Our approaches are based on sequence alignment techniques, taken from the field of bioinformatics, which try to find the optimal alignment between the automaton and the protocol messages. The alignments are then used to further extend the automaton, which represents the syntax rules of the protocol.

The rest of this paper is organized as follows: Section 2 provides some background about sequence alignment algorithms and techniques. The proposed approaches to automata generation through sequence alignment are addressed in Section 3. Section 4 provides some preliminary evaluation results with FTP network traces. In Section 5 we present some related work. And finally, we conclude in Section 6.

2 Sequence Alignment

Sequence alignment is used in bioinformatics to identify similarity regions in sequences of DNA, RNA, or protein. Sequences are represented as a series of nucleotide and amino acid residues that are aligned, based on the residues similarities, within a matrix. Gaps can be introduced in any of the sequences so that similar residues can be aligned in consecutive columns.



(a) Alignment of the two sequences. (b) Substitution matrix.

Fig. 1. Pairwise sequence alignment using dynamic programming.

Figure 1a shows the alignment of two DNA sequences. Vertical lines represent matches between residues, whereas horizontal lines depict gaps. Gaps and mismatches represent the edit operations required to transform one sequence into the other.

2.1 Pairwise Sequence Alignment

Finding the best alignment between two sequences is an optimization problem in which the optimal solution of the overall problem can be deduced from the optimal solutions of many overlapping subproblems. Most sequence alignment algorithms are based on dynamic programming that makes use of such properties by memorizing previously computed subsequence alignments in a substitution matrix. Figure 1b shows the final substitution matrix of a sequence alignment algorithm using dynamic programming. A two-dimensional matrix is constructed with one column for each residue in one sequence and one row for each residue in the other sequence. Thus, if we are aligning sequences of length n and m, the running time of the algorithm is $O(n \times m)$. Each cell in the matrix has the value of the best-aligned subsequence as well as a reference to the adjacent cell from which that value was computed. When the algorithm reaches the last cell of the matrix, it backtraces through the referenced cells.

Usually, alignment algorithms fall into one of two categories, global or local alignment, in which the Needleman-Wunsch algorithm [1] and the Smith-Waterman [2] algorithm are the best examples. Local alignment algorithms try to identify similarity regions within the sequence, as opposed to the global alignment that attempt to align every residue in the entire sequences.

2.2 Multiple Sequence Alignment

Ideally, one would like to align more than two sequences. However, a straightforward dynamic programming algorithm in a k-dimensional matrix (where k is the number of sequences) would be computational unfeasible even for a modest number of sequences—the complexity to align k sequences of length n would be

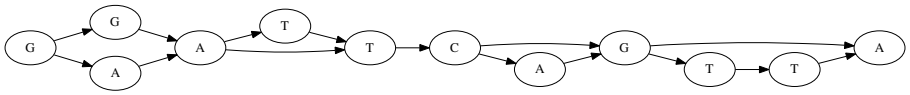


Fig. 2. Partial order graph representing a merged sequence alignment.

$O(2^k n^k)$. Hence, over the last 30 years several approaches have been developed to provide multiple sequence alignment, such as progressive alignment strategies, evolutionary trees, alternative alignment representations, probabilistic alignment, etc., resulting in a very large number of alignment applications, each with their own strengths and weaknesses [3]. Most techniques use heuristics for finding sub-optimal multiple alignments, such as computing all $\binom{k}{2}$ optimal pairwise alignments between every pair of sequences and then combine them together using other heuristics. For instance, the results of multiple sequence alignments can be merged into one reduced consensus sequence that shows which residues are most abundant at each position of the alignment. Clustal, for instance, uses pairwise alignment as a building block for multiple k-way alignments [4]. A phylogenetic tree is built from the similarities between each pair of sequences (pairwise alignment). This progressive multiple alignment heuristic then combines each alignment iteratively as a linear consensus sequence, starting with the closest related groups in the tree towards the root. Consensus sequences, or profiles, allow multiple alignments to be achieved by only performing one pairwise alignment for each new sequence.

2.3 Partial Order Alignment (POA)

Consensus sequences are an incomplete representation of the alignment of multiple sequences. Information about less frequent residues found in early sequence alignments will be lost and cannot be used later when more similar sequences appear. Therefore, the order in which the sequences are aligned is determinant to the quality of the final multiple sequence alignment.

One solution that preserves information of the previous alignments is to represent the consensus sequence as a complete partial-order graph. Partial order alignment (POA) performs multiple sequence alignment by aligning each new sequence (in any order) with a partially ordered graph in which individual sequence letters are represented by nodes, and directed edges are drawn between consecutive letters in each sequence [5]. As each new sequence is aligned, even the less frequent residues are represented, so no information is lost. Thus, instead of a single incomplete consensus sequence, several subsequences are used (one for each chain of consecutive residues). Figure 2 shows the alignment of the sequences from the previous examples, merged into a partial-order graph. As more sequences are aligned, unmatched residues will create new branches. POA extends the dynamic programming method of aligning two sequences by replacing one of the linear sequences with the partial order nodes. Branches in the graph are resolved by grafting a copy of the bifurcations across the matrix, forming additional dynamic programming matrix surfaces that are joined exactly as the individual branches

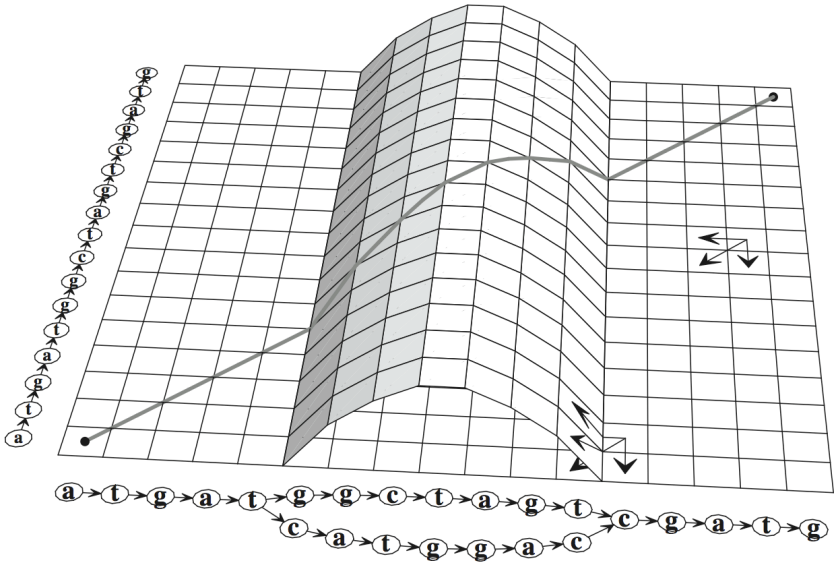


Fig. 3. Substitution graph for the partial order alignment (image taken from [5]).

are joined in the partial order graph. Figure 3 shows the complete substitution matrix for a partial order alignment. The branch in the graph is depicted in the matrix by an alternative sub-matrix, which is then calculated as in the original dynamic programming algorithms.

This method has the advantage of merging similar nodes and sequences into one larger partial order graph, and executing only one pairwise alignment for each new sequence, though all branching subsequences must be visited.

3 Building an Automaton From Protocol Messages

A protocol is a set of rules that dictates the communication between two or more entities. Messages accepted by a given protocol must follow some very specific syntax rules. In this sense, a network protocol can be seen as a formal language whose syntax rules are defined through a formal grammar. In formal language theory, languages are completely deprived of any semantic meaning, and only their syntax formation rules define the words that are accepted. The formal grammar is the set of formation rules that describes how symbols of the alphabet can be combined to form words that are accepted by the language. In another words, the formal grammar of a network protocol describes how bytes can be combined to form acceptable messages.

An equivalent and perhaps more comfortable way of modeling a network protocol is through a finite-state machine automaton. In particular, a deterministic finite automaton is one of the most practical models of computation, since there are trivial linear time, constant-space, and online algorithms to simulate them on

a stream of input. In fact, the automata theory is closely related to formal language theory as the automata are often classified by the class of formal languages they are able to recognize.

The purpose of our work is to provide a solution to construct an automaton that recognizes a specific network protocol based on a sample of its messages. Each state of the automaton represents the sequence of symbols (i.e., bytes of the message) accepted so far, and the remaining symbols required to accept the entire word (i.e., the message of the protocol). Formally, the automata can represented by the 5-tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is a set of all possible states[1], Σ is the alphabet (2^8 possible symbols) of the protocol, δ is the transition function that tells the automaton which state to go to next given a current state and a current symbol (byte), q_0 is the initial state, and F is the set of final states.

Graphs can also be used to represent automata, with nodes denoting states and edges describing the transition functions. To our purposes, representing the automaton as a directed graph or a finite-state machine is equivalent. Graphically, however, the directed graph offers a more clear and simple representation because edges have no labels and each state is represented by the symbol that leads to it.

The partial order alignment (POA) algorithm briefly described earlier, successively aligns new sequences with a growing partial order graph. In order theory, a partial order set formalizes the intuitive concept of an ordered set, which can be represented through a directed acyclic graph (DAG). A DAG is a directed graph in which there are no directed cycles, i.e., in other words, a DAG flows in a single direction where each state can only be visited once.

3.1 Greedy Approach

Our first approach was to extend the POA algorithm, used in DNA multiple sequence alignment, to generate an automaton from protocol messages that would describe the language of the protocol. A pairwise sequence alignment algorithm is used to identify the best automaton path, i.e., the ordered set of nodes that provides a better scoring alignment, and to align the new sequence with the automaton. Figure 4 shows the DAG of the multiple sequence alignments of the previous examples, being aligned with a new sequence. The original pairwise sequence alignment algorithm resorts to a matrix of scores. Instead, our solution avoids matrixes by addressing each column of the matrix as an individual vector of scores, calculated from the adjacent vector of scores. A vector of scores is calculated for each node, so that each position of the vector represents the total score of the best alignment sequence that aligns the symbol of that node with the symbol at that position of the sequence. Additionally, the order in which the vectors are actually calculated is reversed, so that the alignment algorithm starts backtracing the alignment at the first node (instead of the last cell of the matrix). This optimization also relieves the need for additional references (i.e., parent's references on each child).

[1] In the context of protocol reverse engineering, we refer to *state* as the state of the automaton that accepts/rejects network messages, and not the various states that the protocol might have, such as waiting for login, authenticated, etc.

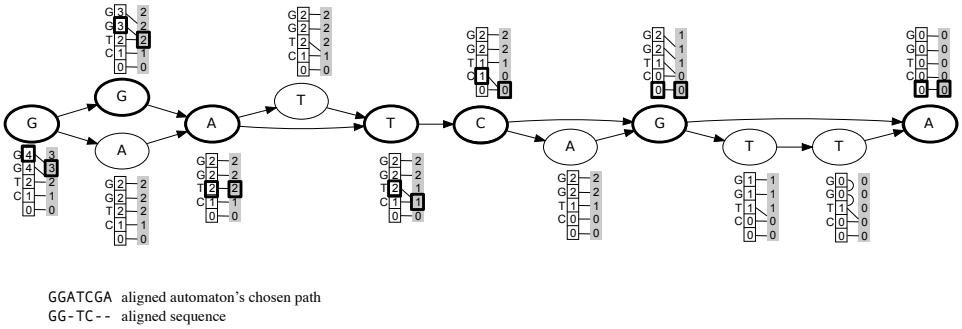


Fig. 4. Aligning a new sequence with the automaton.

This approach uses the Needleman-Wunsch algorithm as a pairwise sequence alignment, but other dynamic programming alignment algorithms, such as the Smith-Waterman, could also be adapted in the same way. The alignment algorithm starts by calculating the vector of scores from the first node, which recursively calculates the adjacent vectors of scores. In the end, the first cell of the first vector of scores provides the remaining cell references for the complete alignment. Pseudocode 1.1 depicts the recursive algorithm to calculate the vector of scores. Each cell of the vector also holds a reference of the adjacent cell from which that value was calculated, i.e., diagonal, right or bottom. The corresponding recurrence (adapted from the Needleman-Wunsch algorithm) for the score $S_{i,j}$ of an optimal alignment between node i and the j-th character of the sequence is given as follows:

$$S_{i,j} = max \begin{cases} S_{i+1,j} + 0 \\ S_{i,j+1} + 0 \\ S_{i+1,j+1} + 1 \end{cases}$$

This score function attributes a score of zero for each additional gap penalty and one for each match. The sequence alignment provides information about the nodes of the automaton that are best aligned with the sequence, i.e., which path is closest to the new message. The maximum value chosen for the vector of scores of node i directs the path of the alignment—in the first case a gap is introduced in the sequence, the second case adds a gap before the next node of the automaton, and in the third case the node i and the j-th character of the sequence are aligned.

The next step is then to incorporate the new sequence in the automaton. The POA algorithm tries to minimize the number of different nodes; hence, it follows a greedy approach by merging every aligned node that shares the same symbol. In the same example, merging the sequence `GGTC` with our automaton would not produce new nodes because all symbols of the sequence were already perfectly aligned in the graph. However, a sequence with a mismatched node, such as `GGTU`, would generate a new edge from `T` (any of the first two, as they would score the same) to a newly created node `U`.

This approach is especially useful to detect common subsequences, wherever they are placed in the messages. For example, consider the IMAP protocol in

```
vectorScores
    Input: Node ← Node of the automaton
    Input: Sequence ← Sequence to be aligned
    Output: Scores ← Vector with the best scores of the children

    // base case
    Scores ← vector of zeros of length #Sequence
    if Node has no children then
        return Scores

    // recursion for every child
    foreach Child of Node do
        Temp ← PairwiseAlignment(Child, Sequence, vectorScores(Child,
              Sequence))
        foreach i of Temp
            if Temp[i] is best than Scores[i] then
                Scores[i] ← Temp[i]
    return Scores
```

Pseudocode 1.1. Algorithm for the calculation of the vector of scores.

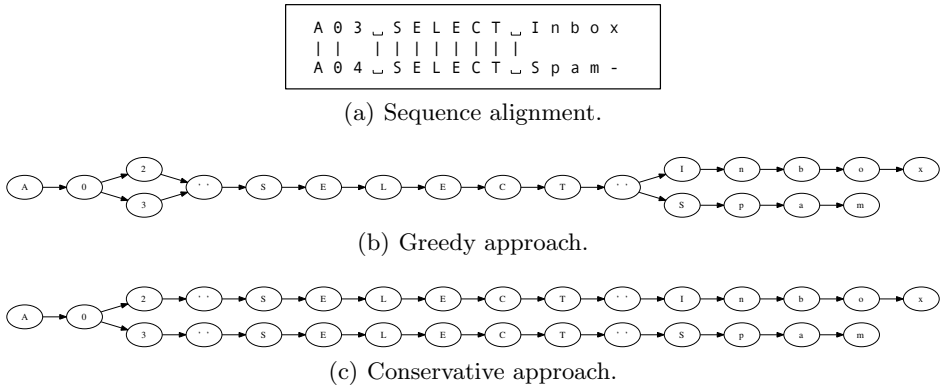
which every command is prefixed with a distinct alphanumeric tag (e.g., `A03 SELECT Inbox`, `A04 SELECT Spam`, etc.). The most relevant part of the message (i.e., the command `SELECT`) appears *after* the variable tag parameter. Hence, similar command names will always be aligned, independently of the remaining of the message. Figure 5b shows the greedy approach while merging the alignment of the two IMAP messages of Figure 5a. The graph evidences the common patterns of both messages, in particular the `SELECT` command.

While messages previously processed are always accepted by the automaton, a greedy merge might be too optimistic and generate transitions allowing messages that do not belong to the language to be accepted. One could imagine a protocol that uses a special prefix to confer different parameters to the same commands, making the syntax of those messages different altogether.

3.2 Conservative Approach

Our other approach to merge the aligned sequences is to produce a conservative automaton, which would *only* accept sequences that were previously aligned, therefore avoiding any false positives. For instance, the automaton of Figure 4, which was constructed from two sequences: `GAATTCAGTTA` and `GGATCGA`, is prone to false positives because it would also accept the sequences `GAATTCGA` or `GGATCAGTTA` (which were never merged before).

In order to keep the automaton strictly correct, i.e., it only accepts messages already received, our conservative approach only branches new nodes when the alignments start to differ (after a common prefix) or ever cease to differ (upon a common suffix). Hence, our solution is to do a greedy merge only for common



(a) Sequence alignment.

(b) Greedy approach.

(c) Conservative approach.

Fig. 5. Two approaches to merge a sequence alignment into the automaton.

prefixes and suffixes, and create new nodes and transitions for the remaining cases. This approach will produce a conservative automaton, with a higher number of states than the greedy one and much more specific (and less permissive) as to which messages it accepts.

Figure 5c shows the conservative approach with two separate paths for the same command `SELECT`, as opposed to the merged path in the greedy approach.

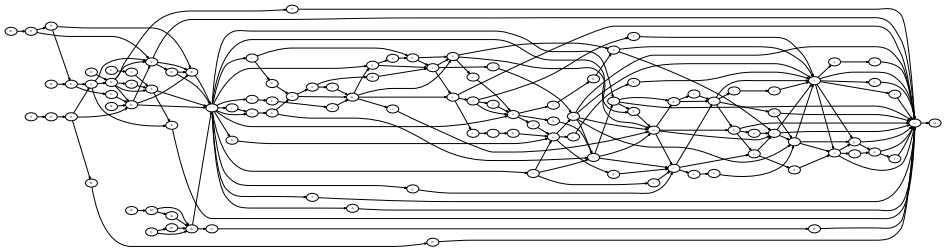
4 Preliminary Evaluation

To evaluate the impact of the conservative and greedy approaches we used the methodology presented here to create two separate automata from the network traces of three FTP sessions. The FTP sessions were composed of 103 messages containing 14 different FTP commands. We then evaluated the resulting automata with respect to the number of nodes and edges of the resulting graphs.

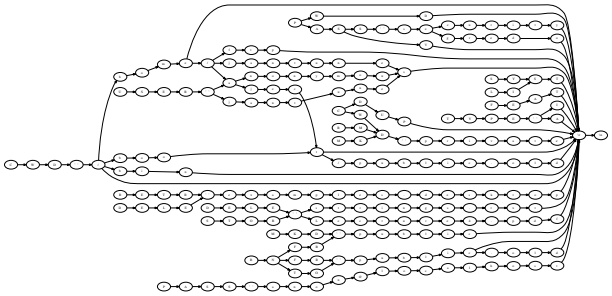
The automata generator prototype was implemented in Java and supports both the greedy and the conservative merging approaches. The program application reads a packet capture file containing the network traces. It filters out all network interactions from other protocols and not coming from the client, so that it only processes FTP messages from the client, i.e., the TCP messages from the FTP session coming from the client. Currently, each network packet is regarded as a single protocol message, disregarding any issues related with the streaming nature of the TCP[2]. The application then iteratively feeds each packet to the automaton, obtaining a sequence alignment, which it will integrate according to the predefined strategy, i.e., greedy or conservative.

This preliminary evaluation tries to reveal the strengths and weaknesses of either approach. Figure 6 shows the resulting directed graphs of the automata. The greedy approach, which optimistically merges every aligned node sharing the same symbol, generated a much smaller automaton with only 110 nodes as

[2] Although TCP is stream-oriented, it is very common that each application-layer message corresponds to a single TCP message, unless the payload is too large.



(a) Greedy approach (110 nodes, 199 edges, 135 branches).



(b) Conservative approach (225 nodes, 242 edges, 32 branches).

Fig. 6. Preliminary evaluation of three FTP sessions with 103 client messages.

opposed to the 225 nodes of the conservative approach. However, 68% of the edges of the greedy automaton are actually branches, thus increasing the complexity of the graph, whereas the conservative automaton the branches account for only 13% of the edges.

A greedy approach will result in smaller but more complex automata, susceptible of false positives. The conservative automata, on the other hand, guarantees that only valid messages are accepted at the cost of some additional separate paths, i.e., several single-transition nodes.

5 Related Work

Protocol reverse engineering is a traditionally manual and tedious process. Aided by network packet analyzers, reverse engineers carefully analyze each packet looking for common and recognizable patterns, such as special delimiters and text fields [6]. The process of reverse engineering a protocol also depends on the number (and quality) of protocol interactions. The more complete the collection of network traces is, more confidence can be placed in the resulting protocol specification.

Since some of the tasks associated with the manual process could be automated or at least automatically assisted, several techniques started to emerge to ease the reverse engineering process. PDB, for instance, is a Protocol Debugger tool that

operates as a transparent network proxy, allowing the operator to set breakpoints on specific packets or events, and inspect and modify individual packets [7].

More recently, new works appeared in the literature trying to infer the protocol specification automatically. Protocol informatics, for instance, applied concepts from bioinformatics to construct a rough description of the protocol [8]. The tool employs sequence alignment algorithms to group similar messages together in phylogenetic trees to automatically identify fields in network packets. The tree is traversed to guide a progressive sequence alignment that will produce a consensus sequence for each cluster of similar messages. As the authors notice, using consensus sequence alone is subject to loss of information as only the most prevalent characters are preserved.

Discoverer is another tool that operates on network traces to perform clustering and type-based sequence alignment to infer message formats [9]. It first tokenizes each message in to binary and text segments to cluster messages of similar format. Clusters are then further divided in a refining process where messages are re-compared and their fields analyzed for cross-field dependencies. This allows some special fields to be identified, such as length fields, or fields whose value differentiates the format of the remaining message. To avoid over-classification, i.e., the creation of small irrelevant clusters, the tool then compares the field structure of the inferred message formats to merge messages with identical format in the same cluster.

Other techniques focus on the binary programs that implement the protocol. Some researches have proposed to use static analysis to generate possible inputs that a program can accept [10]. Beside the fact that it is undecidable to statically determine the complete set of inputs for a program, this approach also suffers from significant scalability issues.

Other researches have also proposed to use dynamic analysis on applications that implement the protocol. Dynamic analysis also circumvents the problems inherent to static analysis, such as memory aliasing and indirect jumps. These tools closely monitor the program's execution while processing the input messages, resorting to dynamic taint analysis to identify the relevant sections responsible for processing and parsing the network packets [11, 12]. One of the limitations of these tools is that they cannot generalize message formats over multiple message samples.

6 Conclusion

In this paper, we have presented two different approaches to build an automaton of a network protocol from network traces. Our solution is based on sequence alignment techniques, taken from the field of bioinformatics, which try to find the optimal alignment between the automaton and the protocol messages. The alignments are then used to further extend the automaton, which represents the syntax rules of the protocol.

The preliminary results of our evaluation with FTP network traces show the main differences in generating a greedy automaton (which tries to minimize the number of nodes with similar symbols) and a conservative one (where new nodes

are created only if they do not introduce any spurious paths). Our results show that a greedy strategy will identify common patterns in the messages, independently of their position, but it will generate more complex automata, i.e., with several bifurcations. On the other hand, the conservative approach will produce simpler automata, but with alternative paths whenever the new message diverges with the automata.

On this work, we have focused on bioinformatics algorithms to generate finite-state machines that would accept protocol messages. The next step towards the protocol reverse engineering will be focused on the refining the automaton and on producing some sort of protocol specification that could be used on other applications, such as intrusion detection systems or fuzzers. We intend to further complement our work with other bodies of research, such as automata reduction, information theory, and formal language theory, to ameliorate and extend the generation of the automata.

References

1. Needleman, S., Wunsch, C.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol **48**(3) (1970) 443–453
2. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology **147** (1981) 195–197
3. Simossis, V., Kleinjung, J., Heringa, J.: An overview of multiple sequence alignment. Current protocols in bioinformatics (2003)
4. Chenna, R., Sugawara, H., Koike, T., Lopez, R., Gibson, T., Higgins, D., Thompson, J.: Multiple sequence alignment with the clustal series of programs. Nucleic acids research **31**(13) (2003) 3497
5. Lee, C., Grasso, C., Sharlow, M.: Multiple sequence alignment using partial order graphs (2002)
6. Beardsley, T.: Manual protocol reverse engineering. BreakingPoint Systems (2009) http://www.breakingpointsystems.com/community/blog/manual-protocol-reverse-engineering.
7. Rauch, J.: PDB: The protocol debugger. In: BlackHat USA, BlackHat (2006)
8. Beddoe, M.A.: Network protocol analysis using bioinformatics algorithms (2005) http://www.4tphi.net/ awalters/PI/PI.html.
9. Cui, W., Kannan, J., Wang, Helen, J.: Discoverer: automatic protocol reverse engineering from network traces. In: Proceedings of the USENIX Security Symposium, Boston, MA, USA (2007) 199–212
10. Newsome, J., Brumley, D., Franklin, J., Song, D.: Replayer: Automatic protocol replay by binary analysis. In: Proceedings of the ACM conference on Computer and communications security, ACM New York, NY, USA (2006) 311–321
11. Caballero, J., Yin, H., Liang, Z., Song, D.: Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In: Proceedings of the ACM Conference on Computer and Communications SecurityC, ACM New York, NY, USA (2007) 317–329
12. Wondracek, G., Comparetti, P., Kruegel, C., Kirda, E., Anna, S.: Automatic network protocol analysis. In: Proceedings of the Annual Network and Distributed System Security Symposium. (2008)

Efficient State Transfer for Recovery-Based Byzantine-Fault-Tolerant State Machine Replication

Rogério Correia and Paulo Sousa

University of Lisboa, Faculdade de Ciências, LaSIGE - Portugal,
rcorreia@lasige.di.fc.ul.pt, pjsousa@di.fc.ul.pt

Abstract. This paper presents an efficient state-transfer protocol for Byzantine-fault-tolerant state machine replication systems enhanced with recovery mechanisms. Usually the recovery of a stateful replica consumes a considerable amount of time, mostly due to state transfer. As a result it is essential to reduce the state transfer time, simultaneously ensuring that correct replicas never lose their state. Our approach consists on creating periodic state checkpoints stored in a distributed secure component, and relying on this component to manage/control state transfer operations. Experimental evaluation results show the performance and overhead of the proposed protocol when combined with a simple application and with a naming and directory service.

Resumo. Este artigo apresenta um protocolo de transferência de estado para sistemas baseados em replicação de máquina de estados tolerante a faltas bizantinas que façam uso de mecanismos de recuperação. Tipicamente, a recuperação de uma réplica com estado consome uma quantidade de tempo considerável, determinada em grande parte pela transferência de estado. Desta forma, torna-se essencial reduzir o tempo de transferência de estado, garantindo ao mesmo tempo que as réplicas correctas nunca perdem o seu estado. A nossa abordagem consiste na criação de salvaguardas de estado periódicas armazenadas num componente seguro distribuído, confiando a este componente a gestão/controlo das operações de transferência de estado. Os resultados de um conjunto de avaliações experimentais mostram o desempenho e o *overhead* do protocolo proposto quando combinado com uma aplicação simples e com um serviço de nomes e directorias.

1 Introduction

Intrusion tolerance [12] has become an increasing area of interest, as the number of techniques to exploit system vulnerabilities has increased, and attacks became more common. One needs to design resilient systems where the exploitation of some vulnerabilities does not result in system corruption. This can be achieved using Byzantine fault-tolerant (BFT) state machine replication (SMR) protocols [7, 2, 6]. These protocols assume that the system operates correctly if at

most f out of the total number (n) of replicas is compromised. These protocols offer added resilience but given a sufficient amount of time, a malicious adversary can find ways to compromise more than f replicas. One interesting way to circumvent this limitation was proposed recently, combining BFT protocols with a proactive-reactive recovery service that rejuvenates replicas from time to time and on-demand, removing the effects of malicious attacks/faults and reducing the probability of the same vulnerability being exploited again [9]. This service assumes a *hybrid* system model and architecture where the system is composed of two parts with distinct properties and assumptions: **payload** and **wormhole** [11]. The *payload* is an *any-synchronous* system in which at most f replicas can be subject to *Byzantine failures* in a given recovery period, and at most k replicas can be recovered at the same time. The *wormhole* is a *synchronous subsystem* connected through a *synchronous and secure control channel*, isolated from other networks, existing one local wormhole per replica.

The proactive-reactive recovery mechanism can be used to provide additional security to a variety of services, namely stateful ones. With the booming of the *Internet*, more and more services are now online and constantly operating over some application state. However, if a service replica is recovered, which typically consists on a shutdown and reboot with the OS and application code loaded from a secure source (since it might have been compromised), an up-to-date state has to be transferred in order to ensure consistency among all replicas. The new state has to be *correct*, in other words, it has to reflect all operations performed by correct service replicas. Yet if the network is early overloaded, the state transfer can be delayed and consequently holding back the entire recovery process. Our main concern is thus conceiving an efficient state transfer mechanism, able to bring service up-to-date in a short amount of time, without overloading the remaining replicas. Some research on state transfer mechanisms [2, 5], has shown how it can be improved using *hierarchal partitions* and *hypervisor-based* technology.

This paper describes WEST, an efficient state transfer protocol for BFT SMR enhanced with *proactive-reactive* recoveries, addressing the issues presented above. This paper follows a previous work on state transfer using wormholes [4]. Our approach builds on the formerly described *hybrid* model, and the idea is to use periodic *checkpoints* (stored in local wormholes) to minimize the amount of state data transferred. Local *wormholes* store a correct state copy (*backup/checkpoint*), which is never lost. *Wormholes* are also responsible for managing and controlling the state transfer process, avoiding possible congestion in the *payload* network.

In the next section, we describe in detail the WEST protocol. Then, Section 3 describes and evaluates the current prototype. Finally, Section 4 concludes the paper.

2 WEST Protocol

This section describes WEST, a wormhole-enhanced state transfer protocol for BFT SMR systems. WEST is a complementary protocol for BFT SMR systems enhanced with a proactive-reactive recovery service. Therefore, the description of WEST assumes the existence of a BFT SMR total order protocol (e.g., [2]) and of a proactive-reactive recovery service, including an API to trigger (reactive) recoveries (e.g., [9]).

2.1 System Model

We assume a hybrid system model and architecture in which the system is composed of *two parts*, with distinct properties and assumptions [11]. These two parts are typically called *payload* and *wormhole*. Applications and the BFT SMR library are assumed to run in the payload part exposed to arbitrary faults and asynchrony. A proactive-reactive recovery service is assumed to run in the wormhole part that is guaranteed to be secure and timely by construction. The WEST protocol has both components that run in the payload and wormhole parts, as it will be explained in Section 2.3. A minimum of $3f + 2k + 1$ replicas are required to ensure availability, on a system where f arbitrary faults may happen between recoveries, with at most k replicas recovering simultaneously [10, 8]. Therefore, we assume a system composed of $n \geq 3f+2k+1$ replicas. Each (payload) replica has its own local wormhole. Local wormholes are connected by a synchronous and secure control channel, isolated from the regular (payload) network that is used to receive, process and answer client requests.

2.2 Description

WEST uses the wormhole part of the system to manage checkpoints, perform garbage collection procedures, and manage the state transfer. The state transfer process is initiated after every recovery (after re-loading the OS), advancing through several steps until its completion. These steps depend on the network characteristics (i.e., if the payload network is constantly receiving requests), the requests logged in the wormhole (saved during the payload replica normal operation) and the checkpoint stored in the wormhole. Figure 1 depicts the system parts and the messages exchanged by them, in a scenario where it is assumed that: there is a stable checkpoint created, the wormhole has requests stored, and there are out-of-date requests. The messages exchanged in other scenarios are illustrated in [3].

The message exchanges depicted in Figure 1 can be defined as a sequence of five conceptual steps described next.

Online: After the replica code being reloaded from a secure source and the OS rebooted, the payload replica sends an ONLINE message to its local wormhole. The client requests received after this point and until the state transfer is completed will be stored in a recovery log for later execution.

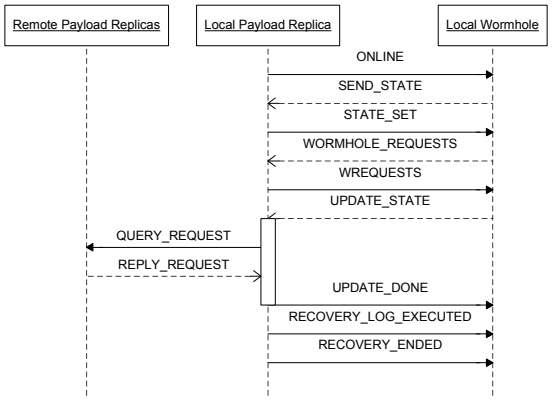


Fig. 1. Messages exchanged during a state transfer scenario.

Set State: The local wormhole sends the latest stable checkpoint to the payload replica (SEND_STATE message).
First requests: Since the wormhole has client requests stored from the period after the creation of the last stable checkpoint and before the payload replica went offline, it sends them to the payload replica for execution (WORMHOLE_REQUESTS message). When these requests have been processed, the replica notifies its local wormhole (WREQUESTS message).
Update state: Next, the local wormhole informs its payload replica to update its state based on the last request seen by the remaining wormholes (UPDATE_STATE message). Once the payload replica receives this message, it queries the remaining replicas for up-to-date requests (QUERY_REQUEST and REPLY_REQUEST messages). Once it gathers all up-to-date requests executes them sequentially in order to update its state.
Finish: Finally, when the replica finishes updating its state (UPDATE_DONE message), it executes its recovery log, which contains the requests received since the last reboot (RECOVERY_LOG_EXECUTED message), informs the wormhole that the state transfer is now complete (RECOVERY_ENDED message), and the payload replica can start processing client requests on behalf of the new state.

2.3 Protocol Internals

As it becomes clear from the description above, the WEST protocol runs in both parts of the system: payload and wormhole. The payload side (Algorithm 1) is responsible for managing client requests, checkpoint creation and state transfer execution. While the wormhole side (Algorithm 2) is responsible for checkpoint management, garbage collection procedures and state transfer coordination. The explanation of both algorithms is presented below.
Request Processing. A client sends a request using BFT SMR total order multicast, which means that requests will be received by all the replicas in the

same order. Each request is then executed (Alg:1, line 2) and logged (Alg:1, line 4) by each local replica and corresponding wormhole (Alg:1, line 5 and Alg:2, line 2). Local wormholes log the requests because this action reduces the amount of data transferred between payload replicas during each recovery, since local wormholes can then deliver these requests once the recovery starts without the need to contact other replicas. Every k requests, a checkpoint creation is triggered (Alg:1, lines 6–7), forcing the payload replica to send its current state to the local wormhole, which then verifies its integrity as we will describe in detail below.

Checkpoint Management. As mentioned before, checkpoints are stored in the secure local wormhole. This action has many advantages, namely it allows to reduce the amount of data transferred when bringing a replica up-to-date, and it guarantees that a stable checkpoint copy that always exist, and cannot be corrupted or lost (as long as no more than f replicas are compromised).

Checkpoints are created periodically after completing a certain number (k) of requests (e.g., $k = 128$). Once the local wormhole receives and stores the checkpoint (Alg:2, line 3), it verifies its integrity by querying the remaining wormholes for the corresponding cryptographic hash of the checkpoint. When the requesting wormhole has received $f + 1$ identical hashes (guaranteeing that at least one hash is from a correct replica), it compares its own checkpoint hash with the receiving ones (Alg:2, line 4). If they match, the checkpoint is stored and labeled as "*stable*" (Alg:2, line 6). If they do not match, the wormhole removes all requests received after the last stable checkpoint and the replica is reactively recovered, as it is counted as one of the faulty ones (Alg:2, line 8). Then, in the following recovery, the state transfer will start from the last stable checkpoint.

As one might notice, there may be times when a local wormhole receives a query to verify the integrity of a checkpoint that it does not possess yet (e.g., because its payload replica is slower than others). In such cases, queries are stored and replied once the requested checkpoint is produced.

Updating State. This phase is the final stage of the state transfer. It is responsible for bringing a recovering replica state up-to-date, since the local wormhole has only the requests it has stored until the replica went offline. The recovering replica needs to fetch (and process) the new client requests received and processed by the remaining replicas while it was offline. Updating state is done in three steps: *finding the last seen sequence number*, *fetching up-to-date requests*, and *executing the recovery log*. We assume that there is a time where the network allows a replica to receive all the requests and the replica processing speed is higher than the request addition rate to the log.

The first step is initiated once the local wormhole receives a message from its replica reporting that the requests stored by the wormhole where successfully executed (Alg:1, line 15). Then the local wormhole starts searching for the last sequence number seen by the remaining local wormholes (as explained before, this sequence number is equal to the replica sequence number since the replica reports it simultaneously to the wormhole and to its local logging) (Alg:2, line 19). This is achieved by requesting the current sequence number to the remain-

ing wormholes, waiting for their reply, and selecting the highest number (H_S) reported by at least $f+1$ wormholes. If and only if, H_S is greater than the local sequence number, the wormhole informs the replica to start fetching requests up to H_S (Alg:2, line 21). This verification avoids fetching redundant requests, i.e., requests that the wormhole already possessed, and therefore already processed by the payload replica in the earlier conceptual phase.

In the next step, the payload replica starts by checking if the recovery log contains the request with sequence number equal to H_S (Alg:1, line 16) in order to fetch only the requests it needs (Alg:1, line 17). This procedure allows the replica to fetch only requests that are not present in the log, reducing the amount of fetched messages and state data. Consider the following example: $H_S = 3000$, current sequence number (seq) = 1500, and the recovery log has the requests with sequence numbers {2000,...,3500}. The number of fetch messages would be reduced to 500 (2000-1500 = 500), because the requests following 2000 are already present in the recovery log. This result contrasts with the original case where 1500 (3000-1500 = 1500) fetch messages would be needed (which is 3 times more), because the requests already present in the log would be ignored, adding not only significant delay to the recovery process but also saturating the network. Once all responses to the QUERY-REQUEST messages are gathered (and selected $f+1$ identical REPLY-REQUEST messages for each request) the requests are executed orderly until seq=H_S (Alg:1, line 27). The update stage is completed if and only if H_S intercepts the recovery log, i.e., if the recovery log has a request with sequence number equal to H_S (which guarantees no gaps between sequential requests). If that is not the case then the payload replica sends a message to the wormhole requesting yet again the maximum seen request, and the process repeats. Note that this iteration eventually stops, since the recovery log is constantly adding new incoming requests, allowing H_S to catch up.

Finally, the last step consists in executing the recovery log (Alg:1, line 28), terminating the state transfer when the last logged request is executed. The recovery log is defined as a concurrent queue data structure, meaning that even while iterating in the log, new requests may (and will be) added to queue's tail, allowing the replica to eventually catch up with the active replicas and ending the state recovery process. Any incoming requests after the log execution are processed as regular requests.

Garbage Collection. A garbage collection protocol is used to prevent message logs and checkpoints stored from growing without bound. Messages and checkpoints are discarded after the wormholes reach a consensus about a safe discard point. This ensures that older messages and checkpoints, necessary for a recovering replica (for verification and fetch purposes) do not get discarded. Due to lack of space, the details on how garbage collection is done can be found in [3].

Algorithm 1 WEST protocol – payload side.

{Variables}
integer $seq = 0$ {Sequence number of the last request received from clients}
Set $R = \emptyset$ {Client requests}
Set $r\_log = \emptyset$ {Recovery log}
bool $recovering = false$ {Indicates if replica is recovering}
data $current\_state = \emptyset$ {The systems current state}
{Parameters}
integer $k = 0$ {Checkpoint period}
integer i {Payload replica id}

{Request Processing}
upon $receive(\langle \text{CLIENT-REQUEST}, r\rangle, c)$
1: **if** $\neg recovering$ **then**
2: $process\_request(r)$
3: $seq = seq + 1$
4: $log\_request(seq, r)$
5: $W\_log\_request(seq, r)$
6: **if** $(seq\ mod\ k) = 0$ **then**
7: $W\_set\_checkpoint(current\_state)$
8: **end if**
9: **else**
10: $r\_log\_request(r, r\_log)$
11: **end if**

{State transfer interface}
when is replica online
12: $current\_state = W\_replica\_online()$
13: $W\_replica\_state\_set()$
service $R\_process\_wormhole\_req(W_r)$
14: $process\_request\_batch(w\_r)$
15: $W\_replica\_update\_payload()$
service $R\_update\_state(latest\_seq)$
16: $delta\_requests = merge\_update(seq, latest\_seq, r\_log)$
17: **for all** r_{id} in $delta\_requests$ **do**
18: $R\text{-}multicast(R, \langle \text{QUERY-REQUEST}, r_{id}, i\rangle)$
19: **end for**
upon $receive(i, \langle \text{QUERY-REQUEST}, r_{id}, j\rangle)$
20: **if** R contains r_{id} **then**
21: $send(j, \langle$ REPLY-REQUEST $, R[r_{id}], i\rangle)$
22: **else**
23: $r = W\_get\_request\_id(r_{id})$
24: $send(j, \langle \text{REPLY-REQUEST}, R[r_{id}], i\rangle)$
25: **end if**
upon $receive(i, \langle \text{REPLY-REQUEST}, r_{id}, j\rangle)$
26: $Fetched \leftarrow Fetched \cup \{r\}$
upon has majority in all fetched requests
27: $process\_fetched\_requests(Fetched, r\_log)$
28: $R\_execute\_rec\_log()$
service $R\_execute\_rec\_log()$
29: $process\_rec\_log(r\_log)$

Algorithm 2 WEST protocol – wormhole side.

```
{Variables}
integer seq = 0 {Sequence number of the last request received from the payload replica}
integer i {Local wormhole id}
Set R = ∅ {Client requests}

{Request Processing}
service W_log_request(new_seq, r)
 1: seq = new_seq
 2: log_request(seq, r)

{State transfer interface}
service W_set_checkpoint(current_state)
 3: chkpId = store_checkpoint(current_state)
 4: chkpOk = verify_checkpoint_integrity(chkpId)
 5: if chkpOk then
 6:    make_stable(chkpId)
 7: else
 8:    purge_checkpoint(chkpId)
 9: end if
service W_replica_online()
10: return last_stable_checkpoint()
procedure W_replica_state_set()
11: req = get_requests_prior_stable_checkpoint(R)
12: validate_ok = validate_wormhole_requests(req)
13: if validate_ok then
14:    R_process_wormhole_requests(req)
15: else
16:    purge_requests(req)
17:    W_replica_update_payload()
18: end if
upon W_replica_update_payload()
19: MaxS = max_seq_seen()
20: if MaxS > seq then
21:    R_update_state(MaxS)
22: else
23:    R_execute_rec_log()
24: end if
```

3 Experimental Evaluation

This section describes experimental evaluation results of the WEST protocol on a LAN. In the first set of experiments, we measured how the state transfer time of a simple application evolves under different (client) request rates. Then, in the second set of experiments, we evaluated what is the performance and overhead

of using WEST in the context of a naming and directory service (NDS).

Prototype. The WEST protocol was implemented in Java and integrated with JBP (Java Byzantine Paxos)[1] [1], a BFT total order multicast protocol also in Java. For the NDS evaluation, two versions of a Java NDS were implemented: a simple replicated NDS service (simple NDS), and a NDS service enhanced with proactive-reactive recoveries that uses WEST to transfer the application state (WEST NDS).

Experimental setup. The results described below were obtained using a total of 7 machines (6 replicas + 1 client). As mentioned in Section 2.1, one needs six $(3f + 2k + 1)$ replicas in order to tolerate one fault $(f = 1)$ between recoveries and to have one replica recovering at the same time $(k = 1)$. In all experiments, the recovery time was set to 150 seconds, representing the time a replica is offline recovering (i.e., before starting state transfer). Finally, all machines used in our experiments were 2.8 GHz Pentium-4 PCs with 2 GBs of RAM running Sun JDK 1.6 on top of Linux 2.6.18, and connected through a Dell gigabit switch.

State transfer performance for multiple request rates and state sizes. In this evaluation the goal was to observe the WEST state transfer performance under multiple client request rates and using different state sizes. Checkpoints were set to be triggered every 30 requests. We integrated WEST in a simple application that only does arithmetic operations. This application is accessed by a single client that works in the following way: it sends a request and waits for its response before sending the next request. The interval between client requests was changed between tests.

Figure 2 shows the results. As expected, the state transfer time increases when the requests rate increases. The slope of the increase is higher for bigger state sizes. When the state size is less or equal than 4KB, the state transfer time is almost immune to the client request rate. When the state size is 1MB, the client request rate has a linear impact on the state transfer time. Finally, if the state size is 4MB, the state transfer time has an exponential increase with more aggressive client request rates. The amount of time consumed in state transfer is related to fetching and processing the requests. For example, for a 4KB state size with a request interval of 20 milliseconds (ms), the number of requests fetched during state transfer is 4687 in comparison to the 1342 requests required for a 100 ms rate. Also, while processing the requests, the replica triggers a checkpoint very often (every 30 requests). So, as the state size increases, this becomes one significant aspect responsible for the delay in the state transfer time.

Impact on NDS performance. In this set of tests, the purpose was to observe the impact that the addition of the WEST protocol has on a NDS, comparing to simple NDS that does not use WEST. This was done by measuring the service latency (delay between requests and replies) with and without WEST. In practice we should note a decrease in the system performance, since WEST adds extra

[1] Avaliable at http://www.navigators.di.fc.ul.pt/software/jitt/jbp.html

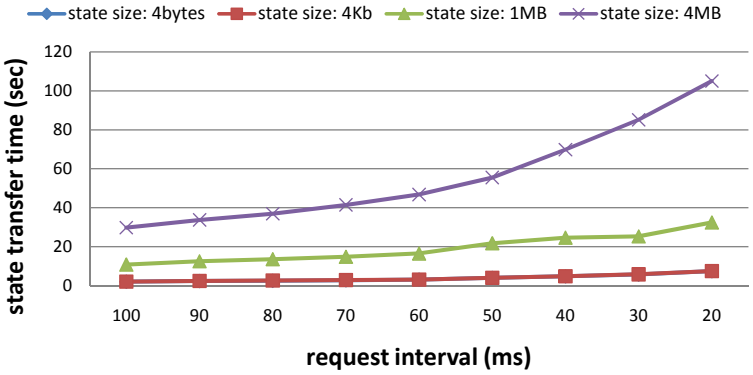


Fig. 2. Average duration of state transfer under different request rates and state sizes. Checkpoint period (k) = 30 requests.

messages to the network. For this test we fixed the client request rate for both simple NDS and WEST NDS to 50 ms. The checkpoint period for the WEST NDS was set to 128, as it seems to be the appropriate value for this client request rate [3]. Figure 3 presents the average latency of both simple NDS and WEST NDS.

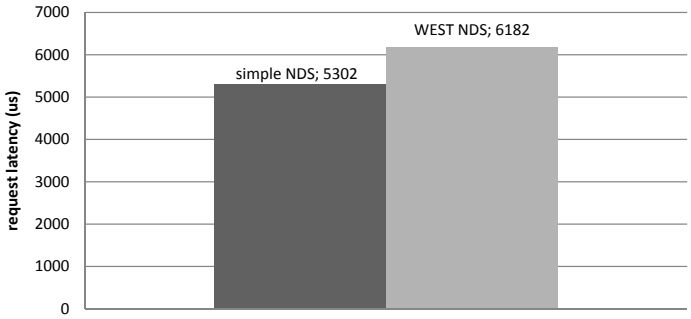


Fig. 3. Average latency of client requests for simple NDS and WEST NDS. Checkpoint period (k) is set to 128, the interval between client requests is set to 50 ms.

As we can observe, the WEST NDS latency is slightly higher but still close to the simple NDS. This increase is related to a few factors mentioned above, namely the extra messages introduced by WEST to control the state transfer and to fetch missing requests. The simple NDS registered an average of 5302 microseconds (μs) while the WEST NDS presented a minimum latency of 5867 μs and an average of 6182 μs. We can conclude that WEST increased the request latency from 10% to 15%, which is a good value given the (fast) client request

rate choosed (50 ms).

State transfer performance when using the NDS. In this final set of tests, the goal was to evaluate the amount of time it takes for the NDS to recover its state when using WEST. The client request rate was set to 50 ms, and the checkpoint period to 128. Table 1 presents the average, maximum, and minimum state transfer time.

	Average	Maximum	Minimum
State transfer time (microseconds)	6641	6728	6367

Table 1. WEST state transfer time while recovering the NDS state. Checkpoint period (k) is set to 128 and the interval between client requests is set to 50 ms.

The WEST NDS state transfer time varied between 6367 ms and 6728 ms, with an average of 6641 ms. This value is slightly higher than the one observed in Figure 2. In this figure, for a 4KB state size (which is approximately the state size of the NDS state), the state transfer time was roughly 4 seconds, which is about 40% less comparing to the transfer time of the NDS state. The only significant factor that changed between tests was the application state and the operations made over this state. So we can conclude that the increase in the state transfer time is closely related to the application semantics: in the NDS case, the exhausting number and type (read/search and write) of operations executed on the NDS.

Analysis of the results. Results show us that one of the factors that increase the state transfer time is the client request rate. However, choosing an appropriate checkpoint period, can help to reduce the state transfer time significantly [3]. On the other hand, adding WEST to a NDS Java application has a negligible impact. The experimental results showed that WEST NDS decreased the system performance from 10% to 15%, and that the state transfer time of a typical NDS state (4KB) is about 6 seconds.

4 Conclusions

In this paper we proposed WEST, an efficient state-transfer protocol for Byzantine fault-tolerant (BFT) state machine replication (SMR) systems enhanced with proactive-reactive recovery. The WEST approach consists on creating periodic state checkpoints stored in a distributed secure component (wormhole), and relying on this component to manage/control state transfer operations. This approach allowed us to provide extra security to the state transfer process (since the wormhole stores a stable and incorruptible copy of the state) and to decrease the state transfer time, by reducing the amount of requests needed to update a recovering replica. When combined with a naming and directory service, WEST

overhead was negligible, given that it has a small impact on the service latency, and recovers the application state in a few seconds, with no system downtime.

Acknowledgments

This work was partially supported by FCT through the Multiannual Funding and the CMU-Portugal Programs.

References

1. A. N. Bessani, E. P. Alchieri, M. Correia, and J. da Silva Fraga. DepSpace: A Byzantine fault-tolerant coordination service. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Systems Conference - EuroSys 2008*, Apr. 2008.
2. M. Castro and B. Liskov. Practical Byzantine fault-tolerance and proactive recovery. *ACM Transactions Computer Systems*, 20(4):398–461, Nov. 2002.
3. R. Correia. *WEST: Wormhole-Enhanced State Transfer*. Master thesis, Department of Informatics, University of Lisbon, July 2009.
4. R. Correia and P. Sousa. WEST: Wormhole-enhanced state transfer. In *Proceedings of the DSN 2009 Workshop on Proactive Failure Avoidance, Recovery and Maintenance (PFARM)*, Estoril - Portugal, 2009.
5. T. Distler, R. Kapitza, and H. P. Reiser. Efficient state transfer for hypervisor-based proactive recovery. In *Proceedings of the 2nd Workshop on Recent Advances on Intrusion-Tolerant Systems (in conjunction with Eurosys 2008, Glasgow, Scotland, April 1, 2008)*, 2008.
6. R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles - SOSP'07*, Oct. 2007.
7. F. B. Schneider. Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, Dec. 1990.
8. P. Sousa. *Proactive Resilience*. PhD thesis, Department of Informatics, University of Lisbon, May 2007. DI/FCUL TR-07-11.
9. P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo. Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Transactions on Parallel and Distributed Systems*, 2009. to appear.
10. P. Sousa, N. F. Neves, and P. Verissimo. Resilient state machine replication. In *PRDC '05: Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing*, pages 305–309. IEEE Computer Society, 2005.
11. P. Verissimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1), 2006.
12. P. Verissimo, N. F. Neves, and M. P. Correia. Intrusion-tolerant architectures: Concepts and design. In *Architecting Dependable Systems*, volume 2677 of *LNCS*. 2003.

Construção Observável de um Quorum de Nós Não-Sybil na Vizinhança Rádio de uma Rede Ad Hoc Sem Fios ⋆

Diogo Mónica, João Leitão, Luís Rodrigues, and Carlos Ribeiro

INESC-ID / IST
{diogo.monica, joao.c.leitao, ler, carlos.ribeiro}@ist.utl.pt

Resumo O ataque Sybil é uma ameaça importante à operação segura e confiável das redes ad hoc sem fios. Propomos um algoritmo para a construção de um quorum parcialmente coerente de nós não sybil, na vizinhança local de uma rede sem fios. O algoritmo é baseado na combinação de diferentes tipos de testes de recursos, de forma a garantir não só a detecção e posterior exclusão das identidades sybil, e também garantir a eficiência na construção do quorum. O algoritmo garante que todos os nós correctos no sistema possuem um quorum válido.

Abstract The sybil attack is a relevant treat to the secure operation of wireless ad hoc networks. We propose a novel algorithm to construct a partially consistent sybil-free quorum on a wireless one-hop neighborhood. The algorithm is based on the combination of distinct resource tests that ensure the detection and removal of sybil identities, and the efficiency of the quorum construction. Our algorithm ensures that all correct participants in the system own a valid quorum.

1 Introdução

Os quorums têm vindo a ser utilizados como um mecanismo para aumentar a disponibilidade e eficiência de vários serviços em sistemas distribuídos [1,2]. Tipicamente nestes sistemas, existe um conjunto de servidores que disponibiliza o mesmo serviço de forma replicada a um conjunto de clientes. De forma a garantir a operação correcta destes sistemas, qualquer operação deve ser realizada sobre uma fracção de réplicas, de forma a que todas as operações se interceptem num número suficiente de réplicas (tipicamente, o necessário para mascarar a existência de réplicas incorrectas). Em redes sem fios, a possibilidade de um nó malicioso usar múltiplas identidades torna a construção e utilização de um quorum uma tarefa não trivial.

A utilização de múltiplas identidades por um nó malicioso é designada na literatura por "Ataque Sybil". Este ataque consegue comprometer a integridade dos sistemas baseados em quorums, assim como de um conjunto de protocolos distribuídos; e.g. armazenamento, encaminhamento, agregação de dados, votação, detecção de intrusões e partilha de recursos [3,4]. Embora tenham sido propostas várias soluções para impedir ou mitigar este ataque [5,6,4], a maior parte delas requer a existência de uma entidade centralizada confiável ou a existência de segredos pré-partilhados, sendo sua utilização impraticável em redes ad hoc expontâneas, e.g. sem pré-configuração [7].

⋆ Este trabalho foi parcialmente suportado pela FCT com a bolsa PTDC/EIA/65588/2006 e pelo LEMe pelo projecto WiMesh.

Uma das técnicas que pode ser usada para combater o ataque Sybil em redes ad hoc passa pela utilização de testes aos recursos. Estas soluções funcionam partindo da premissa de que é possível estabelecer um limite à quantidade de recursos disponível em cada nó. A verificação desta premissa num conjunto de identidades permite verificar se existem no sistema mais identidades do que nós.

Este artigo propõe um algoritmo de criação de um quorum de tamanho q, constituído por identidades não sybil, o Quorum Não-Sybil (QNS), numa vizinhança rádio de uma rede ad hoc sem fios (i.e. uma rede com um único salto). No final da execução do algoritmo, todos os nós correctos possuem um quorum constituído por uma maioria de identidades utilizadas por nós correctos, e que se intercepta globalmente em $q - f$ destas identidades. O tamanho do quorum final é um parâmetro do algoritmo. Por exemplo, se existirem f nós bizantinos na vizinhança de um nó, poderá ser necessário ter um quorum de tamanho q igual a $3f + 1$, de forma a que este seja tolerante a falhas bizantinas. A existência de um quorum permite a delegação, num conjunto de nós correctos, de decisões críticas relativamente à operação da rede. O algoritmo QNS utiliza uma combinação de diferentes tipos de testes de recurso, de forma a tonar a sua execução mais eficiente e robusta.

O resto do artigo está organizado da seguinte forma. Na Secção 2 é introduzido o modelo e enunciado o problema a resolver. A Secção 3 descreve a nossa aproximação ao problema. Na Secção 4 abordamos a correcção da solução apresentada. Finalmente, na Secção 5 concluímos o artigo, fornecendo algumas direcções para trabalho futuro.

2 Modelo de Sistema e Enunciado do Problema

2.1 Modelo de sistema

Neste artigo consideramos uma rede sem fios síncrona, ponto-para-multiponto, composta por N nós. Adicionalmente, fazemos as seguintes hipóteses sobre o funcionamento do sistema.

Nós Correctos e Nós Bizantinos Um nó pode ser correcto ou bizantino (não-correcto). Um nó bizantino pode exibir um comportamento não previsível, como por exemplo o envio de mensagens arbitrárias. Os nós bizantinos podem também estar em conluio entre si. No máximo existem $f < N$ nós bizantinos no sistema.

Sincronismo Assumimos que o sistema é síncrono e que o tempo é modelado como uma sequência de passos independentes. Em cada passo, todos os nós executam uma quantidade limitada de computação, e podem enviar ou receber uma mensagem de/para a rede. Em cada passo pode ser enviada no máximo uma mensagem em cada um dos canais de comunicação disponíveis. Se num determinado passo mais do que dois nós enviarem uma mensagem no mesmo canal rádio, assumimos que ocorre uma colisão.

Recursos Limitados Assumimos também que todos os nós, quer sejam correctos ou bizantinos, possuem apenas um dispositivo de rádio e uma capacidade de computação limitada. Os dispositivos de rádio podem operar em qualquer um de K canais disponíveis, não podendo no entanto, comunicar em mais do que um canal simultaneamente. Finalmente, limitamos também o número de colisões intencionais ci, que um nó bizantino pode provocar numa sequência de passos de dimensão P. Neste cenário assumimos que nós correctos têm oportunidade não nula de comunicar se a condição

$ci \cdot f < P$ for verdadeira (os nós correctos podem, no entanto, gerar também colisões não intencionais)[1].

Comunicação Os nós comunicam através do envio de mensagens numa reda partilhada, sem fios, numa única vizinhança rádio. Assumimos que todas as mensagens são enviadas em modo de difusão, num canal único, e recebidos por todos os outros nós que se encontrem na vizinhança rádio a escutar esse canal. Salvo informação explícita em contrário, toda a comunicação entre os nós é realizada através de um canal único. Como se verá, os restantes canais serão utilizados para realizar testes de recurso rádio. Assumimos também que os canais de transmissão são confiáveis: quando não há colisão, as mensagens são entregues sem perdas ou corrupção dos dados. Adicionalmente, assumimos que sempre que uma colisão ocorre no meio sem fios, todos os nós, incluindo os emissores da mensagem, são capazes de detectá-la. Este objectivo pode ser atingido através da utilização de um mecanismo de reforço de colisão: Após a detecção de uma colisão, os nós correctos transmitem por tempo suficiente para garantir que todos os nós que transmitiram as mensagens que causaram a colisão estão cientes de que esta ocorreu [8]. Quando ocorre uma colisão, nenhum nó é capaz de receber uma mensagem. Finalmente, os recursos limitados dos nós impedem que o meio de comunicação seja alvo de ataques de negação de serviço, como por exemplo, através da transmissão contínua de sinal para o meio.

Nós e identidades Todas as mensagens enviadas por um nó estão associadas a uma *identidade*. O receptor de uma mensagem consegue sempre associar uma mensagem recebida com a sua identidade de origem, excluindo-se a possibilidade de falsificação da origem das mensagens. Esta propriedade pode ser garantida através da utilização de criptografia de chave pública, e fazendo com que cada nó assine digitalmente todas as mensagens enviadas com a chave associada à identidade. Note-se que a utilização de criptografia de chave pública não implica a utilização de um PKI: pares de chaves assimétricas pública-privada podem ser geradas aleatoriamente, sem necessitar da intervenção de terceiros, e utilizadas como identidades.

Um nó pode usar uma ou mais identidades para comunicação. Representamos como *ids(n_i)* o conjunto de identidades utilizadas por um nó n_i. De forma inversa, representamos como *usa(id)* o conjunto de nós que utilizam uma certa identidade *id*. Um nó correcto utiliza sempre uma única identidade que não é usada por mais nenhum outro nó na rede (correcto ou bizantino). De forma inversa, os nós bizantinos podem tentar utilizar mais do que uma identidade para comunicação. Estas múltiplas identidades são chamadas identidades sybil. Assim, para identidades utilizadas por nós correctos, temos que $\{id_i\} = ids(n_i)$ e $\{n_i\} = usa(id_i)$. Não existe limite para o número de identidades $|ids(b)|$ que um nó bizantino b pode utilizar. Adicionalmente, e uma vez que os nós bizantinos podem estar em conluio, a mesma identidade pode ser utilizada por múltiplos nós bizantinos. Consequentemente, se *bid* é uma identidade utilizada por um nó bizantino, temos $1 \leq |usa(bid)| \leq f$.

2.2 Enunciado do Problema

O nosso objectivo é retornar um quorum de identidades QNS_i, em cada nó correcto n_i, cujo tamanho alvo (e máximo) é dado pelo parâmetro q. As identidades em cada QNS_i

[1] Note-se que esta hipótese baseia-se numa outra hipótese mais genérica, de que os nós têm limites à capacidade de transmissão

podem pertencer a nós correctos ou bizantinos. Repare-se que, uma vez que os nós bizantinos podem deixar de operar em qualquer momento, o quorum final resultante pode conter menos do que q identidades, contendo no mínimo $q - f$. Assim, a intercepção do quorum retornado em cada nó correcto, contem pelo menos $q - f$ identidades pertencentes a nós correctos. Mais precisamente, o problema pode ser definido através das seguintes propriedades:

Quorum Parcialmente Coerente e sem Sybils Existe um conjunto de identidades de nós correctos (QNS), comum a todos os QNS_i, tal que $q \geq |QNS| \geq q - f$, com uma probabilidade arbitrariamente perto de 1. Defina-se $\Gamma()$ como:

$$\Gamma(n_i) = \begin{cases} 1, \text{ if } n_i \text{ é um nó correcto} \\ 0, \text{ caso contrário} \end{cases} .$$

Seja $QNS = QNS_1 \cap QNS_2 \cap ... \cap QNS_{N-f}$ o conjunto de identidades comuns a todos os QNS_i dos nós. Então,

$$\sum_{n_i}^{usa(QNS)} \Gamma(n_i) \geq q - f.$$

Terminação Probabilista Todos os nós correctos retornam o quorum com uma probabilidade arbitrariamente próxima de 1, num número finito de passos.

O tamanho alvo do conjunto QNS (q), é um parâmetro do nosso algoritmo. Repare-se que não é possível impedir que os nós bizantinos façam parte do quorum. Caso um nó bizantino se comporte de forma correcta, não é possível distingui-lo de um nó correcto, e consequentemente, pode vir a fazer parte do quorum final. Assim, o parâmetro q tem de ser escolhido de acordo com o tipo de aplicações para o qual o quorum será utilizado. Tipicamente, q terá um valor que assegure que o número de nós bizantinos não consegue influenciar a aplicação para a qual o quorum está a ser usado (e.g.: $q = 3f + 1$ [9]).

3 Construção do Quorum não-Sybil

O algoritmo de construção do quorum não-Sybil (QNS) utiliza três fases distintas, a saber: fase de geração do *nonce*; fase de selecção de candidatos; e a fase de validação do quorum. O esqueleto do algoritmo encontra-se representado na Figura 1. Os parâmetros do algoritmo serão descritos mais à frente.

No centro do algoritmo está a fase de selecção de candidatos (fase 2). O objectivo desta fase é encontrar um conjunto adequado de identidades, de tamanho σ, para formar o quorum não-sybil. Este conjunto é obtido recorrendo a um *teste de recursos computacional* (TRC). Para cada identidade que pretende propor, um nó necessita de resolver um problema que lhe consumirá tempo de processador. Atendendo aos recursos limitados que cada nó possui, este teste limita o número de identidades que cada nó pode propor. De modo a evitar a utilização de soluções calculadas previamente, o problema possui como parâmetro um nonce desconhecido até ao momento da sua utilização. Este nonce é obtido através da combinação da contribuição de n identidades distintas (n é outro dos parâmetros do algoritmo QNS). Este é o propósito da primeira fase do algoritmo. Como se verá, é extremamente difícil recorrendo apenas a um TRC assegurar que não existem entidades Sybil na lista de candidatos. Por este motivo, o algoritmo

```
algorithm QNS is
    C ⟵ ∅; //conjunto de identidades candidatas ao quorum
    nonce ⟵ null; // string, nonce para o TRC
    nonce ⟵ nonceGeneration(n); // Primeira fase
    C ⟵ candidateSelection (nonce); // Segunda fase
    C ⟵ quorumValidation (C); // Terceira fase
    return C;
```

Figura 1. Esqueleto do algoritmo de construção do QNS (executado em cada nó i).

possui uma terceira fase, que recorre a um *teste de recursos rádio* (TRR), e que possui por objectivo eliminar identidades sybil do conjunto de candidatos e provar a todos os restantes elementos da rede que as identidades do quorum são, de facto, usadas por, pelo menos, um igual número de nós distintos.

3.1 Geração do Nonce

O objectivo da fase de geração do nonce é a criação de uma sequência de dígitos aleatória, que depende das contribuições de n identidades distintas. Nesta fase, o algoritmo itera através de uma série de passos de comunicação, onde em cada passo um nó ou tenta contribuir para o nonce, ou apenas escuta outras contribuições. Quando um ou mais nós tentam contribuir para o nonce no mesmo passo, é gerada uma colisão. Como foi descrito previamente, no caso da existência de uma colisão, todos os nós (incluindo os emissores da mensagem que colidiu), detectam a colisão e descartam as contribuições, garantido a coerência do nonce em todos os nós correctos.

O procedimento da geração do nonce é simples. São recolhidas contribuições dos participantes e colocadas num conjunto, pela ordem pela qual foram recebidas. Este procedimento termina assim que o conjunto atinge um tamanho alvo n. O algoritmo encontra-se representado na Figura 2. Inicialmente, o conjunto de contribuições para o nonce é vazio. Em cada passo, os nós que ainda não conseguiram adicionar a sua contribuição, podem tentar difundi-la através do envio de uma mensagem que contem o identificador do nó, e um valor aleatório. Para evitar colisões desnecessárias, os nós apenas transmitem com uma probabilidade p_t. Caso não seja gerada nenhuma colisão devido a esta transmissão, todos os nós adicionam a contribuição do emissor ao nonce.

```
algorithm nonceGeneration (n) is
    nonceSet ⟵ ∅
    while (|nonceSet| < n) do
        if id_i ∉ nonceSet and rand() < p_t then
            rndvalue ⟵ rand();
            if broadcast.send ([id_i, rndvalue]) ≠ collision then
                nonce ⟵ nonce ∪ [id_i, rndvalue];
        else
            msg ⟵ broadcast.receive();
            if msg ≠ null and msg ≠collision then
                if source(msg) ∉ ids(nonce) then
                    nonce ⟵ nonce ∪ msg;
    return hash(values(nonce));
```

Figura 2. Geração do nonce (executado em cada nó i).

Os nós que, num determinado passo, não tentarem adicionar o seu identificador, vão escutar o meio de forma a receberem contribuições dos outros nós. Se um nó recebe uma contribuição de qualquer outro participante, adiciona-a ao conjunto de contribuições. Este passo não gera incoerências, uma vez que assumimos que no evento de uma colisão todos os nós recebem a mensagem correctamente. É possível que um nó que esteja à escuta não receba nenhuma contribuição. Isto pode acontecer ou porque nenhum nó decidiu difundir a sua identidade nesse passo, ou devido à existência de uma colisão.

Este mecanismo é suficiente para garantir que, desde que haja mais do que n nós no sistema, todos os participantes correctos vão convergir para o mesmo conjunto de geração do nonce. O facto dos nós possuírem recursos limitados, assegura que se n for suficientemente grande, pelo menos um nó correcto consegue contribuir para este conjunto. No final, todos os nós retornam o mesmo valor de forma determinista, recorrendo à utilização de uma função de síntese criptográfica (e.g., a SHA-1 [10])).

O valor de n deve garantir que pelo menos um nó correcto consegue participar na geração do nonce. Assumindo que um nó malicioso pode transmitir ci mensagens em P passos, n deve respeitar: $ci \cdot f < n \leq P$. Note-se que com esta condição este mecanismo termina, de forma correcta, num número finito de passos, visto que eventualmente n nós correctos terão a oportunidade de transmitir.

3.2 Selecção de Candidatos

O objectivo da fase de selecção de candidatos é obter um conjunto de σ identidades que são candidatos para formar o quorum de q identidades não-sybil. Para evitar que os nós bizantinos proponham múltiplas identidades sybil para este conjunto de candidatos, esta fase recorre à utilização de um teste de recurso computacional (TRC). A ideia passa essencialmente por fazer com que os nós sejam obrigados a resolver um puzzle criptográfico antes de conseguirem propor uma identidade. Este puzzle recebe como parâmetros o nonce gerado na fase anterior, e a identidade a ser proposta. Assim, esta fase de selecção de candidatos consiste em: receber e validar as propostas que são enviadas assim que outros nós resolvem o puzzle criptográfico; e resolver o puzzle criptográfico, para tentar fazer parte dos candidatos.

Este procedimento assume que o tempo médio para efectuar o teste de recurso computacional (i.e., para resolver o puzzle criptográfico), é muito superior ao tempo necessário para disseminar (e validar) todas as propostas dos σ nós. Desta forma, no momento em que um nó bizantino consegue acabar o cálculo do puzzle criptográfico para uma segunda identidade (sybil), já todos os nós correctos foram capazes de difundir as suas propostas (legítimas).

Repare-se que caso um nó bizantino conheça o nonce *a priori*, é capaz de pré-calcular tantas soluções quantas necessário, de forma a conseguir propor um qualquer número arbitrariamente grande de identidades sybil ao sistema. Daqui a importância da fase de geração do nonce.

A Figura 3 apresenta o pseudo-código para a fase de selecção de candidatos. O algoritmo usa duas tarefas. Uma das tarefas é utilizada para resolver o puzzle criptográfico, e retornar a solução numa variável t. A outra tarefa é utilizada para receber as respectivas propostas dos outros participantes ou, caso a solução do puzzle já tenha sido calculada, para propor a sua identidade. Uma mensagem de PROPOSAL, inclui a identidade que está a ser proposta e a respectiva solução para o puzzle criptográfico. Quando os nós recebem uma proposta, validam a solução do puzzle. Caso seja correcta, todos os nós adicionam a identidade correspondente ao conjunto de candidatos.

```
algorithm candidateSelection(nonce,q) is
    C ⟵ ∅;  done ⟵ 0;  t ⟵ 0;
    σ ⟵ ; // Número de propostas necessárias
    Task 1: // Resolução do crypto-puzzle
        a_i ⟵ trc.solve(nonce, id_i);
        t ⟵ 1;
    Task 2: // Verificação da resposta
        while |C| ≤ σ | timeout do
            if t = 1 and rand() < p_t and done = 0 do
                if broadcast.send(id_i,a_i) ≠ collision do
                    C ⟵ C ∪ {id_i};
                    done ⟵ 1;
            else
                m ⟵ broadcast.receive();
                if m = (PROPOSAL) do
                    if trc.verify(nonce,m.id, m.a) do
                        C ⟵ C ∪ {id};
        return C;
```

Figura 3. Fase de selecção de candidatos (executado em cada nó i).

Idealmente, um TRC perfeito não permitiria que um participante resolvesse um segundo puzzle a tempo de propor mais que uma identidade para a lista de candidatos. No entanto, a natureza destes testes faz com que o tempo de resolução do puzzle possua uma distribuição aleatória, existindo uma probabilidade não-zero de um nó bizantino propor mais que uma identidade. No entanto, assumindo que existe um limite $s > 0$ ao número total de identidades sybil que os nós bizantinos conseguem propor no TRC, podemos obter o número de respostas necessárias σ, de forma a conseguir garantir que são propostas pelo menos $q - f$ identidades correctas no pior caso, $\sigma \geq q + s$. O limite s é possível de demonstrar, podendo o leitor interessado encontrar mais detalhes em [11].

Teste de Recurso Computacional Existem vários formas de concretizar um TRC. A utilização de TRCs foi inicialmente proposta em [12] como um método de defesa contra e-mail não desejado ("spam"), e mais tarde usada como um método de defesa contra ataques de negação de serviço em [13,14]. Em [15], os autores usam um TRC para verificarem se os participantes de um sistema possuem a quantidade expectável de poder de computação. Este teste, intitulado crypto-puzzle, consiste em forçar as identidades participantes do sistema a resolver um problema criptográfico, solúvel apenas por cálculo de força-bruta, num certo limite de tempo. Desta forma, um nó com restrições de poder computacional, tem um limite ao número de puzzles criptográficos que consegue resolver num certo período de tempo, garantido assim um limite superior no número de identidades sybil que consegue apresentar ao sistema. No nosso algoritmo, assumimos que o teste de recurso computacional tem o seguinte interface: Um nó tem de invocar *trc.solve(nonce,id)* para resolver o puzzle criptográfico, usando um nonce e uma identidade como parâmetros de entrada; a invocação retorna a resposta ao puzzle. Existe também um método *trc.verify(nonce, id,* t_{id}*)* que permite qualquer nó validar uma resposta ao puzzle criptográfico. Assumimos que uma resposta pode ser verificada num único passo.

Exemplo de Puzzle Criptográfico Embora o nosso algoritmo não esteja limitado a um puzzle criptográfico específico, damos o exemplo de um puzzle criptográfico que

cumpre os nossos requisitos. Dada uma função de síntese criptográfica $\mathcal{H}$, e um operador $\mathbf{left}_w(str)$, que retorna uma sub-sequência composta pelos w dígitos mais à esquerda da sequência str, o puzzle criptográfico pode ser definido da seguinte forma (t_{id} é a solução para o puzzle criptográfico, w um parâmetro estático de configuração do puzzle, que controla a *dificuldade*):

$$trc.solve(nonce, id) = \{t \in \Re : left_w(\mathcal{H}(nonce||id||t)) = 0^w\}$$

$$trc.verify(nonce, id, t) = \begin{cases} true, \ \mathbf{left}_w(\mathcal{H}(nonce||id||t)) = 0^w \\ false, \ \mathbf{left}_w(\mathcal{H}(nonce||id||t)) \neq 0^w \end{cases}$$

De forma a resolver o puzzle criptográfico, o nó tem de encontrar a sequência de dígitos t, tal que o valor resultante da aplicação da função de síntese criptográfica à concatenação do nonce, a identidade e t, é uma sequência de dígitos onde os w dígitos mais à esquerda são zero.

O algoritmo conhecido mais rápido para a resolução de uma colisão parcial numa função de síntese criptográfica é o calculo através de força bruta [16]. Note-se que o puzzle não é interactivo, uma vez que o nonce torna-se globalmente conhecido na fase anterior do nosso protocolo, e id é a identidade para a qual o nó se encontra a calcular a resposta. Esta não-interactividade é importante, porque faz com que o puzzle criptográfico seja globalmente confiado, e não esteja associado um par de nós específicos.

A verificação da validade da resposta ao puzzle pode ser facilmente obtida através do calculo de uma operação de síntese criptográfica da resposta transmitida por um qualquer nó, e da verificação de existência de zeros nos primeiros w dígitos da sequência resultante. Além disso, este puzzle, em particular, é publicamente verificável, uma vez que pode ser verificado de forma independente por qualquer participante do sistema, sem necessidade de possuir um segredo.

3.3 Validação do Quorum

A fase de validação do quorum é capaz de detectar identidades sybil num conjunto de identidades C, através de um teste de recurso rádio. Este teste é baseado na hipótese que nenhum nó tem a capacidade de transmitir simultaneamente em dois canais distintos. Ao obrigar os utilizadores de cada identidade a transmitir num canal diferente ao mesmo tempo, impede-se um nó bizantino de defender mais do que uma identidade. Infelizmente, como os nós que pretendem validar essa identidade também não podem ouvir em mais do que um canal ao mesmo tempo, o teste tem que ser repetido várias vezes para que com elevada probabilidade se detecte um nó que pretenda defender mais do que uma identidade. Por outro lado se o número de nós a ser testado for superior ao número de canais disponíveis é necessário testar todas as combinações de nós, pelo que o teste tem que ser repetido várias vezes.

O método "trr.lenght (C)" retorna o número de passos necessários para testar $|C|$ identidades com K canais rádio, para que com elevada probabilidade se detecte as identidades sybil. O método "trr.schedule(C)" retorna uma lista de identidades que são supostas transmitir em cada passo do teste. Assumimos que os canais são atribuídos às identidades do 1 até ao canal h, em ordem crescente.

Esta fase do algoritmo é representada na Figura 4, e executa-se durante um período fixo de passos, definidos pela função "trr.lenght()". Em cada passo, o nó verifica se é suposto transmitir. No caso afirmativo, transmite uma mensagem de VALIDATE no canal

especificado pelo agendamento do teste de recurso rádio. Caso contrário, escuta em um dos canais de rádio disponíveis, aleatoriamente. Note-se que os nós que não pertencem ao conjunto de candidatos, escutam sempre, em todos os passos. Caso seja detectado silêncio no canal (não houver transmissão), a identidade que estava a agendada para transmitir nesse canal é adicionada à lista de exclusão.

Esta fase termina com o retorno de no máximo q identidades da seguinte forma. Todos os nós ordenam, por ordem lexicográfica, as identidades validadas pelo teste de recurso de rádio. Este conjunto é concatenado com uma ordenação similar das identidades excluídas pelo teste. Do conjunto resultante cada nó selecciona as primeira q identidades, e esse conjunto é considerado como o quorum final QNS.

Note-se que todos os nós retornam um conjunto com no mínimo $q - f$ identidades ainda que que cada nó possa retornar um conjunto diferente, uma vez que nós bizantinos podem defender diferentes identidades Sybil em fases diferentes do TRR. As propriedades do teste de recurso de rádio, e a ordenação dos conjuntos, garantem que o quorum retornado em cada nó correcto respeita a propriedade de "Quorum Parcialmente Coerente e sem Sybils" (Secção 2.2). Iremos discutir esta propriedade na Secção 4.

Teste de Recurso Rádio O teste de recurso rádio (TRR) é um tipo particular de um teste de recursos. Um TRR assume que cada um dos nós tem acesso a apenas um único dispositivo de rádio, e utiliza limitações conhecidas destes dispositivos. Por exemplo, se os dispositivos de rádio forem incapazes de transmitir simultaneamente em frequências diferentes, este facto pode ser explorado. Assim como a maior parte das soluções de teste de recursos, os TRR têm o potencial de suportar protocolos que não necessitem de pré-configuração ou segredos pré-partilhados. Os TRRs têm ainda uma vantagem adicional comparativamente com os outros tipos de testes de recursos; tanto quanto sabemos, são o único tipo de testes de recursos que permite que identidades não participantes num teste específico consigam avaliar, e validar os resultados do teste, não necessitando para isso de confiança em nenhum dos participantes.

Em [17], analisámos a complexidade de vários TRRs diferentes. Mostrou-se que o custo de correr sistematicamente TRRs de forma a detectar identidades sybil, numa população com muitas identidades, é proibitivamente caro em termos de mensagens, limitando a escalabilidade das soluções que o façam. Esta observação motiva a necessidade de explorar métodos alternativos de aproveitar as vantagens dos TRRs, evitando o custo de testar cada uma das identidades que participam na rede, permitindo melhorar o desempenho destas soluções.

TRR Utilizado O algoritmo apresentado na Figura 4 pressupõe que são testadas todas as combinações das $\mathcal{C}$ identidades, K a K. Para além disso, como referimos, para cada combinação, é necessário que os nós transmitam passos suficientes para que a a probabilidade de detecção de identidades Sybil seja tão grande quanto necessário. Por razões de espaço, é impossível reproduzir aqui os cálculos que permitem mostrar o valor de *trr.lenght($\mathcal{C}$)* em função da probabilidade de detecção alvo. Estes resultados, assim como uma descrição do escalonamento dos testes, podem ser encontrados em [17].

4 Correcção do Algoritmo

Nesta secção apresenta-se um rascunho das provas da correcção do protocolo, de acordo com as propriedades defendidas anteriormente.

```
algorithm quorumValidation (C) is
    excluded ⟵ ∅;
    I ⟵ trr.schedule(C);
    for (j = 0 to trr.length (C)) do
        if id_i ∈ I[j] then
            broadcast.send (I[j].indexOf(id_i), VALIDATE);
        else
            channel ⟵ rand(1,K);
            if broadcast.receive (channel) = null then
                excluded ⟵ ∪ {I[j][channel]};
    return [sort(C \ excluded)];
```

Figura 4. Procedimento de Validação do Quorum (executado em cada nó i).

Lema 1: O nonce é gerado com a participação de um nó correcto, com uma probabilidade arbitrariamente próxima de 1.
Esboço da prova: Esta propriedade é garantida pela limitação dos nós bizantinos em efectuarem transmissões, e pelo valor n de contribuições necessárias para gerar o nonce. □

Lema 2: Assumindo a existência de um nonce correcto, o conjunto de candidatos resultante do teste de recurso computacional contêm no mínimo $q - f$ identidades pertencentes a nós correctos, com uma probabilidade arbitrariamente próxima de 1.
Esboço da prova: Uma vez que o algoritmo de selecção de candidatos termina quando forem atingidas as $\sigma = q + s$ identidades, e uma vez que com uma probabilidade arbitrariamente alta, os nós bizantinos apenas conseguem propor $f + s$ identidades, então as $(q - f)$ identidades restantes pertencem necessariamente a nós correctos. □

Lema 3: Nenhum identificador pertencente a um nó correcto é eliminado pelo teste de recurso de rádio por outro nó correcto.
Esboço da prova: Assuma-se o contrário, e que o identificador de um nó correcto é eliminado por um nó correcto devido ao teste de recurso de rádio. Nesse caso, o nó que propôs essa identidade ou omite um passo de comunicação devido a uma falha, ou usa esse passo para defender uma outra identidade. Em qualquer dos casos esse nó não é correcto - uma contradição. □

Lema 4: O teste de recurso de rádio, com uma probabilidade arbitrariamente próxima de 1, limita nos quorums de todos os nós correctos o número de identidades propostas por nós bizantinos a um máximo de f, para valores de f inferiores a K.
Esboço da prova: Esta é uma propriedade do teste de recurso de rádio que deriva directamente das limitações dos rádios e do desenho deste teste. Um conjunto de nós f apenas pode transmitir simultaneamente em f canais de rádio distintos, indicando que simultaneamente apenas podem defender f identidades. A melhor estratégia para um nó nestas condições é defender sempre a mesma identidade, o que resulta na presença de, no máximo, f identidades pertencentes a nós bizantinos nos quorum. □

Como descrito na Secção 2, pretendemos que o nosso protocolo possua as propriedades de "Quorum Parcialmente Coerente e sem Sybils" e "Terminação Probabilista". De forma a provar a propriedade "Quorum Parcialmente Coerente e sem Sybils", provamos primeiro os seguintes Lemmas:

Lemma 5 (Sem Sybils): Com uma probabilidade arbitrariamente perto de 1, não há identidades sybil no QNS_i de uma identidade correcta n_i. Mais precisamente, seja:

$U_i = \bigcup_{id \in QNS_i} usa(id)$. Temos que $|U_i| \geq |QNS_i|$.
Esboço da prova: Esta propriedade deriva directamente do Lema 4. Assuma-se o contrário, e que existe uma identidade sybil no QNS_i de um nó correcto n_i. Uma vez que na última fase do QNS todas as identidades participam em testes de recurso rádio, a existência de uma identidade sybil implicaria que o TRR não eliminou todas as identidades sybil do conjunto de candidatos. No entanto, e de acordo com o Lemma 4, o resultado do TRR não contêm identidades sybil, com uma probabilidade arbitrariamente perto de 1, o que contradiz a hipótese. □

Lemma 6 (Do no harm): Seja Q_i o resultado do TRR de um nó correcto n_i. Caso haja uma identidade correcta A em Q_i, então, a identidade A também está presente em todos os resultados da fase de validação de todos os outros nós correctos. Mais precisamente, se $A \in Q_i$ para o nó n_i, então $A \in Q_i \ \forall_{n_i}$.
Esboço da prova: Este Lemma deriva directamente do Lemma 3, e do facto de a fase de selecção de candidatos retornar o mesmo conjunto de identidade $\mathcal{C}$ em todos os nós. Assuma-se o contrário, e que existe uma identidade A de um nó correcto que está presente no resultado da fase de validação do quorum Q_i de um nó correcto n_i, mas não no resultado Q_j de um outro nó correcto n_j. No entanto, e tendo em conta que o conjunto de entrada da fase de validação é o mesmo em todos os nós ($\mathcal{C}$), e que o TRR não elimina identidades de nós correctos (Lemma 3), se $A \in Q_i$ então $A \in Q_j$, o que contradiz a hipótese. □

Com estes dois Lemmas conseguimos agora provar a propriedade "Quorum Parcialmente Coerente e sem Sybils" do QNS.

Teorema 1 (Quorum Parcialmente Coerente e sem Sybils) Existe um conjunto de identidades de nós correctos (QNS), comum a todos os QNS_i, tal que $q \geq |QNS| \geq q-f$, com uma probabilidade arbitrariamente perto de 1, tal como definido na Secção 2.2.
Esboço da prova: Este Teorema deriva dos Lemmas 2, 3, 5 e 6. De acordo com o Lemma 2, o resultado da fase de selecção de candidatos ($\mathcal{C}$), contem pelo menos $q - f$ identidades correctas. Considere-se ainda que: i) o conjunto $\mathcal{C}$ é o mesmo para cada nó; ii) o TRR não elimina identidades de nós correctos (Lemma 3), e todas as identidades correctas estão presentes no resultado de cada nó correcto (Lemma 6); iii) existem no máximo f identidades de nós bizantinos em cada QNS_i de cada nó correcto (Lemma 5). Assim, e se ordenarmos por ordem alfabética o resultado do algoritmo QNS, e limitarmos o conjunto às primeiras q identidades, temos a garantia que: o conjunto resultante é composto por, no máximo, f identidades de nós bizantinos e, no mínimo, pelas primeiras $q - f$ identidades de nós correctos, do conjunto ordenado do NSQ_i. □

Teorema 2 (Terminação Probabilista): Todos os nós correctos retornam o quorum com uma probabilidade arbitrariamente próxima de 1, num número finito de passos.
Esboço da prova: Para cada uma das fases do protocolo, existe um número finito de passos que assegura as suas propriedades com elevada probabilidade (como descrito na Secção 3). O número total de passos necessário para terminar o protocolo, é a soma dos passos consumidos em cada fase. □

5 Conclusões e Trabalho Futuro

Neste artigo foi proposto um algoritmo para construir um quorum parcialmente coerente composto apenas por identidades não-sybil numa rede sem fios de vizinhança local.

O algoritmo é baseado no uso de testes de recursos, não só para detectar (e excluir) identidades sybil, mas também para optimizar o processo de construção do quorum. Foram também enunciadas e verificadas algumas propriedades que o algoritmo possui. Como trabalho futuro, fica a expansão da solução para redes com vários saltos.

Referências

1. Malkhi, D., Reiter, M.: Byzantine quorum system. Distributed Computing **11** (1998) 569–578
2. Malkhi, D., Reiter, M., Wool, A.: The load and availability of byzantine quorum systems. In: SIAM Journal of Computing. (1997) 249–257
3. Douceur, J.R.: The sybil attack. In: IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, London, UK, Springer-Verlag (2002) 251–260
4. Newsome, J., Shi, E., Song, D., Perrig, A.: The sybil attack in sensor networks: analysis & defenses. In: Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on. (2004) 259–268
5. Yu, H., Gibbons, P., Kaminsky, M., Xiao, F.: Sybillimit: A near-optimal social network defense against sybil attacks. In: Security and Privacy, 2008. SP 2008. IEEE Symposium on. (May 2008) 3–17
6. Yu, H., Kaminsky, M., Gibbons, P.B., Flaxman, A.: Sybilguard: defending against sybil attacks via social networks. SIGCOMM Comput. Commun. Rev. **36**(4) (2006) 267–278
7. Pirzada, A.A., McDonald, C.: Establishing trust in pure ad-hoc networks. In: ACSC '04: Proceedings of the 27th Australasian conference on Computer science, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2004) 47–54
8. Fullmer, C.L., Garcia-Luna-Aceves, J.: Fama-pj: A channel access protocol for wireless lans. In: Proc. ACM Mobile Computing and Networking '95, ACM (1995) 76–85
9. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems **4** (1982) 382–401
10. Eastlake, D.E., Jones, P.E.: Us secure hash algorithm 1 (sha1). http://www.ietf.org/rfc/rfc3174.txt?number=3174
11. Mónica, D., Leitão, J., Rodrigues, L., Ribeiro, C.: Observable non-sybil quorum construction in one-hop wireless ad hoc networks. Technical Report 34, INESC-ID (June 2009)
12. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, London, UK, Springer-Verlag (1993) 139–147
13. A. Juels, J.B.: Client puzzles: A cryptographic countermeasure against connection depletion attacks. In: NDSS '09: Proceedings of NDSS '99 (Networks and Distributed Security Systems). (1999) 151–165
14. Back, A.: Hashcash - a denial of service counter-measure. Technical report (2002)
15. Aspnes, J., Jackson, C., Krishnamurthy, A.: Exposing computationally-challenged Byzantine impostors. Technical Report YALEU/DCS/TR-1332, Yale University Department of Computer Science (July 2005)
16. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology **13** (2000) 361–396
17. Mónica, D., Leitão, J., Rodrigues, L., Ribeiro, C.: On the use of radio resource tests in wireless ad hoc networks. In: Proceedings of the 3rd Workshop on Recent Advances on Intrusion-Tolerant Systems (WRAITS), Estoril, Portugal (June 2009) F21–F26

Uma arquitectura para um serviço de encaminhamento seguro em redes de sensores sem fios

Pedro Amaral, Henrique Domingos

Departamento de Informática, FCT/UNL
CITI – Centro de Informática e Tecnologias da Informação
Pedro-amaral@netcabo.pt, hj@di.fct.unl.pt

Resumo. Este artigo apresenta uma solução para um sistema de encaminhamento seguro para RSSF que apresenta serviços de segurança para protecção das comunicações entre os sensores e tolerância a intrusões com base em mecanismos de resiliência activa baseada esquemas probabilísticos de descoberta e auto-organização da rede, de selecção e estabelecimento de rotas, de reorganização topológica da rede e de refrescamento de chaves e segredos criptográficos. A implementação e o estudo do protocolo foram feitos com base num ambiente de simulação de redes de sensores sem fios e comunicações por rádio IEEE 802.15.4 e tem em vista a possível utilização em redes que podem comportar milhares de nós, com estrutura multi-hop e organização orientada para grupos, formando estes grupos hierarquias de processamento, encaminhamento e controlo de disseminação de dados.

Palavras-chave: Redes de Sensores sem Fios (RSSF), Encaminhamento Seguro em RSSF, Tolerância e Resistência Activa a Intrusões

1 Introdução

Redes de de sensores sem fios (RSSF) são redes de comunicação por radiofrequência baseadas na norma IEEE 802.15.4 [1] ou variantes [15]. Em geral são redes auto-organizadas, constituídas por dispositivos (sensores) de baixo custo, de pequenas dimensões, dotados de capacidades computacionais limitadas, com recursos de comunicação reduzidos e com importantes restrições quanto à energia que podem utilizar, podendo esta ser finita. Normalmente os sensores medem, pré-processam e transmitem valores associados à monitorização de fenómenos físicos, funcionando de forma autónoma ou sem intervenção humana. Os valores podem ser pesquisados ou podem ser encaminhados até nós especiais de captura e processamento de dados (designados por *sync-nodes*) ou podem ser enviados, através de *gateways*, para outros sistemas, onde são finalmente processados e analisados Os princípios de organização e estruturação de protocolos para RSSF são bastante diversos dos que orientam a concepção de protocolos e pilhas de serviços nas redes de computação convencionais. O mesmo se aplica aos mecanismos e serviços de segurança.

Os ataques contra RSSF podem ter um grande impacto no funcionamento correcto dessas redes, podendo afectar com maior ou menor facilidade desde o nível físico ou de ligação de dados [16] até aos diversos níveis de estruturação de uma arquitectura de software e suas aplicações. Operando estas redes de forma autónoma e em ambientes hostis (ou fracamente vigiados), as condições de

segurança de operação tornam-se particularmente difíceis de garantir. As dificuldades são ainda mais óbvias face a adversários que possam actuar por intrusão física a zonas mais ou menos alargadas da rede ou que possam capturar e provocar intrusões nos sensores, podendo mesmo replicar comportamentos incorrectos na rede. Ao nível do encaminhamento de dados, um ataque pode forçar o tráfego a fluir por um nó controlado pelo atacante, comprometendo as diversas fases de funcionamento do sistema de encaminhamento, nomeadamente: (i) descoberta e organização da topologia da rede, (ii) anúncio, selecção e estabelecimento de rotas e (iii) manutenção das rotas de encaminhamento [4].

Muitos dos protocolos de encaminhamento propostos na investigação das RSSF não contemplam medidas de segurança. Apenas algumas propostas estudaram contra-medidas face a possíveis tipologias de ataques. De entre estes últimos, muito poucas consideram modelos de adversário com hipóteses de actuação por intrusão na rede e que possam assim quebrar a segurança pelo comprometimento de segredos ou parâmetros criptográficos guardados ou partilhados pelos sensores.

Neste artigo propõe-se um serviço de encaminhamento seguro para RSSF que resista a ataques externos ao meio de comunicação e que apresente características de resistência a intrusões com base em mecanismos de resiliência activa. Pretende-se que sistema proposto possa operar em RSSF de grande escala, de estrutura *multi-hop* e auto-organizadas em topologias orientadas para grupos. As medidas de segurança presentes no protocolo são materializadas por uma pilha de serviços de segurança que contém mecanismos de criptografia para protecção das comunicações por rádio e mecanismos intrínsecos inspirados em métodos probabilísticos, que permitem estabelecer critérios de resiliência activa para prevenir ou minimizar o impacto de ataques por intrusão que tenham como alvo o encaminhamento correcto de dados através da rede. A referida pilha foi concretizada para estudo num ambiente de simulação experimental para RSSF [8] com capacidade para simular redes constituídas por milhares de nós. Para completude do estudo foram implementados e adicionados, ao ambiente do simulador, diversos componentes complementares para: avaliação de consumo de energia, estudo de condições de cobertura da rede, indicadores de fiabilidade, mecanismos de reconfiguração topológica e de refrescamento de chaves criptográficas bem como componentes de avaliação da resistência da rede face aos anteriores indicadores, no caso de simulação de ataques por intrusão.

Organização do artigo. As restantes secções do artigo estão organizadas do seguinte modo: a secção 2 enquadra a tipologia de ataques e modelos de adversário que inspiram a concepção do sistema de encaminhamento proposto. A secção 3 apresenta a arquitectura de software que está na base da solução de encaminhamento seguro proposta. O serviço de encaminhamento e seus mecanismos são apresentados na secção 4. A secção 5 é dedicada a aspectos de implementação da solução. A secção 6 reporta a avaliação da implementação com base em alguns dos testes de simulação realizados. Finalmente, a secção 7 apresenta as principais conclusões e aborda aspectos de investigação futura.

2. Ataques ao encaminhamento em redes de sensores sem fios

A abordagem do sistema de encaminhamento seguro proposto neste artigo considera dois tipos de ataques que podem ter lugar nuam RSSF: ataques externos e ataques internos. Os ataques externos são essencialmente ataques à comunicação rádio entre um ou mais sensores correctos. Os ataques internos são ataques realizados por intrusão, com possível alteração do comportamento correcto dos nós (sensores) da rede bem como captura de informação guardada nos mesmos.

Os ataques externos são pois ataques do tipo *man-in-the-middle*, com base nas hipóteses de um modelo de adversário do tipo Dolev-Yao[2]. Estes adversários podem produzir ataques por acções ilícitas de acesso ao meio de comunicação, tendo em vista a leitura, modificação, reordenação, forja, descarte, reprodução ou inserção de mensagens, em qualquer ponto da rede.

Os ataques internos podem ser efectuados por intrusão que pode ser feita a partir da captura física e comprometimento de sensores legítimos, com modificação maliciosa do seu ambiente de execução ou acesso indevido a segredos (ou chaves criptográficas) mantidos nos sensores violados. Estes ataques colocam em causa a segurança das comunicações baseada em mecanismos e protocolos criptográficos convencionais que dependam da manutenção segura desses segredos. Ataques internos bem sucedidos podem ainda permitir replicar comportamentos maliciosos do nó comprometido e, possivelmente, tirar partido da localização dessas réplicas para produzir ataques que alterem, manipulem e controlem a topologia da rede e, assim, obter o controlo de parte mais ou menos significativa da mesma.

A maioria dos ataques ao encaminhamento tem por base a ideia de se provocar uma situação em que nós maliciosos estejam contidos em rotas anunciadas, seleccionadas, estabelecidas e mantidas na rede. Deste modo, o adversário conseguirá atrair e interceptar os pacotes ou impedir que esses prossigam até ao destino, podendo alterar indevidamente os seus conteúdos. Alguns destes ataques estão apresentados e conceptualizados como modelos de hipóteses de adversários [3] e [4], nomeadamente: falsificação de informação de anúncios de encaminhamento, *wormholes, sinkholes, sybil atacks, blackholes, HELLO flood ou rushing based atacks*. Detalhes sobre a caracterização e tipologia destes ataques pode ser encontrada na bibliografia indicada, constituindo as hipóteses possíveis de um modelo de adversário para efeitos do presente trabalho.

3. Pilha de segurança e mecanismos de resiliência previstos

De forma a estruturar uma solução que cubra a anterior tipologia de ataques ao encaminhamento e de modo a estender as defesas a mecanismos de resiliência face a potenciais intrusões, concebeu-se uma pilha de segurança que se encontra dividida essencialmente em 2 camadas principais: uma camada inferior que fornecesse primitivas de comunicação segura (ao nível MAC, protegendo as comunicações 802.15.4). Esta camada garante uma primeira linha de defesa face a ataques externos, disponibilizando comunicação básica segura às camadas superiores.

A camada superior da pilha de segurança permite endereçar a problemática da resistência a intrusões, através de mecanismos de base probabilística para estabelecer critérios de resiliência pró-activa. Estes critérios, vistos como princípios de recuperação activa face a intrusões [5], estabelecem o princípio de que os mecanismos de prevenção, tolerância ou recuperação face a intrusões devem ser integrados no processamento normal e permanente do sistema, sem que existam intervalos de inactividade ou de recuperação de estados correctos de execução, no qual estariam activos apenas os mecanismos de detecção e correcção de falhas. Para estabelecimento dos critérios de resiliência activa ao nível do encaminhamento, consideram-se no protocolo de encaminhamento os seguintes mecanismos selecção probabilística de multi-rotas usadas com redundância aleatória, mecanismos de reorganização topológica automática da rede e mecanismos de refrescamento de chaves partilhadas para garantir propriedades de segurança passada e futura perfeitas.

4. Serviço de encaminhamento seguro

O funcionamento base do protocolo está dividido em 3 fases distintas, descritas à frente: (1) auto-organização segura da rede, (2) selecção e estabelecimento de rotas em segurança e (3) manutenção segura do encaminhamento de dados. Na organização da rede cada nó pode estar a cumprir uma de várias tarefas distintas, que obrigam a diferentes tipos de participação no protocolo: nó líder de grupo, nó encaminhador ou nó comum. Um nó que esteja a cumprir a tarefa de líder de grupo tem como objectivos a formação de um grupo (que filiará outros nós), permitindo a esses nós enviarem mensagens detectadas como eventos pelo líder de grupo. Um nó encaminhador, que esteja filiado num grupo, tem como principal função o reencaminhamento de eventos detectados no grupo, até estes atingirem o nó sink. Um nó comum filiado num grupo terá um papel passivo no encaminhamento, apenas enviando eventos. O papel dos diversos nós, no momento da inicialização da rede, é escolhido probabilisticamente pelos próprios nós, seguindo o esquema do protocolo que será apresentado de seguida.

Para o funcionamento das várias fases e mecanismos do protocolo serão necessários dois padrões de comunicação: comunicações *unicast*, de qualquer nó da rede até ao nó sink e comunicações *broadcast*, do nó sink para os nós da rede e que suportará o envio de mensagens de manutenção da rede a partir do nó sink.

Todas as comunicações na rede são protegidas pela camada de segurança básica anteriormente referida, utilizando-se mecanismos criptográficos baseados em métodos criptográficos simétricos e funções de síntese segura de mensagens para se assegurar requisitos de confidencialidade, autenticidade (por utilização de assinaturas do tipo HMAC), integridade das mensagens e protecção contra retransmissão ilícita de mensagens, protegendo-se assim as comunicações de ataques externos.

No sistema estabelecem-se e utilizam-se diferentes tipos de chaves criptográficas: chaves de rede, chaves partilhadas, chaves de grupo e chaves para comunicações *broadcast*.

A chave de rede é uma chave comum para toda a rede e serve para comunicações que não necessitem de condições de autenticidade, para descoberta

de nós ou para estabelecimento inicial de outras chaves como por exemplo a chave de grupo, usada depois para comunicações seguras dentro de cada grupo. A chave de rede é pré-distribuída por todos os nós antes de serem dispostos na área geográfica da rede.

As chaves partilhadas são usadas por qualquer nó da rede que queira estabelecer comunicação mutuamente autenticada com outro (sendo chaves *pair-wise*). Com base nas chaves partilhadas o nó consegue-se autenticar perante outro nó, pois apenas os dois nós em causa possuirão essa chave partilhada. As chaves partilhadas são pré-distribuídas anteriormente, utilizando-se um esquema de estabelecimento aleatório a partir de um conjunto de chaves pré-instaladas, sendo o esquema baseado no propcesso random pairwise-key pre-distribution scheme[6]. Deve notar-se que todos os nós da rede partilham uma chave deste tipo com o nó sink.

As chaves de grupo são utilizadas em comunicações com garantias de autenticação local dentro de um grupo. Assim, qualquer nó correcto filiado num grupo poderá iniciar a comunicação assegurando aos nós receptores que pertence ao grupo. As chaves de grupo permitem obter melhores garantias de autenticação face à chave de rede. Apenas os nós correctamente filiados num grupo irão possuir a chave correcta usada nesse grupo. A utilização desta chave permite também obter melhorias de desempenho no envio de mensagens para os elementos do grupo e permitem o envio de mensagens autenticadas, confidenciais e íntegras com um único envio. As chaves de grupo são geradas e distribuídas (ou refresadas) pelos nós líderes de grupo e são inicialmente estabelecidas durante a fase de auto-organização da rede, que será apresentada mais à frente.

A chave criptográfica a ser utilizada exclusivamente para comunicações *broadcast* é criada e gerida segundo o mecanismo inspirado no sistema SPINS ou baseado no esquema Micro-Tesla [7].

Fase de auto-organização da rede em grupos. O protocolo pressupõe inicialmente uma organização topológica da rede em grupos. A partir dessa organização faz-se posteriormente a formação de rotas usadas para o envio *multi-hop* de mensagens. A formação dos grupos é feita de forma probabilista. Cada nó irá tentar-se ligar a todos os grupos com os quais conseguir estabelecer contacto, optando, aleatoriamente, por seleccionar um dos grupos como grupo principal, o qual irá servir para o envio de mensagens associadas aos eventos monitorizados. Cada nó pode no entanto manter grupos secundários que serão usados como alternativos a uma possível perca de confiança em sensores do grupo principal ou em caso de falha de cobertura no grupo principal. A fase de auto-organização da rede em grupos tem o seu começo quando os nós da rede decidem, com base num parâmetro probabilístico, iniciar um grupo propondo-se como líderes de grupo. O restante esquema de auto-organização resume-se do seguinte modo:

1. Auto-nomeação (probabilística) como líderes de grupo
2. Envio de mensagem a anunciar novo grupo (anúncio repetido durante um determinado período de tempo)
3. Recepção de mensagens de novo grupo (no caso de partilhar uma chave com o emissor). Envio das mensagens de pedido de entrada no grupo.

4. Recepção da mensagem de pedido de entrada no grupo. Envio da mensagem com a chave de grupo gerada pelo líder.
5. Recepção de chave de grupo. Escolha como nó encaminhador ou nó comum. No caso de escolher tarefa de encaminhador, enviar mensagem ao grupo com essa notificação

Note-se que os passos 3, 4 e 5 são executados de um modo assíncrono entre os vários nós que pedirem acesso a um grupo, sendo no entanto mantida a sequência apresentada.

Fase de selecção e estabelecimento de rotas. Durante esta fase são constituídas as rotas que permitem o envio de mensagens de qualquer nó da rede para o nó *sink*. Para contornar a problemática da cobertura e não particionamento da rede foram pensados dois tipos de rotas: as rotas principais (estabelecidas a partir do nó *sink*) e as rotas secundárias (estabelecidas entre nós fronteiros aos grupos).

Estabelecimento de rotas directas. As rotas directas permitem alcançar o nó sink apenas por comunicação entre nós encaminhadores de diferentes grupos. Assim, todos os *hops* de uma rota directa destinam-se a nós encaminhadores, pertencendo todos a diferentes grupos. Isso permite alcançar o nó sink de uma forma mais eficiente a nível energético e potencialmente mais seguro.

A selecção e estabelecimento de rotas directas até ao nó sink tem início no próprio nó sink, ao enviar uma mensagem que permite alertar a existência de uma rota até ele. A construção das várias rotas passa pela inundação dessa mensagem pela rede, concatenando, sucessivamente, a actual rota na informação contida na mensagem, de forma a poderem ser evitados ciclos na criação das rotas. O esquema de estabelecimento das rotas directas resume-se da seguinte forma:

1. Envio de mensagem a informar nova rota (originalmente vinda do nó *sink*)
2. Recepção da mensagem com nova rota, caso partilhe uma chave com o emissor. Avaliação da rota recebida. Envio de pedido de confirmação de rota para verificar a bi-direccionalidade da ligação.
3. Recepção da mensagem com pedido de confirmação. Envio da mensagem com confirmação da rota.
4. Recepção da mensagem de confirmação de rota. Concatenação do seu identificador à rota recebida. Recomeçar do passo 1.

Paralelamente ao processo descrito, de inundação das mensagens de novas rotas, através da sua recepção e interpretação por parte dos nós encaminhadores, também os nós líderes de grupo irão aproveitar a disseminação dessas mensagens para inferir sobre a cobertura do respectivo grupo e armazenar as rotas disponíveis no seu grupo. Para o efeito as mensagens que contêm as diferentes rotas serão recebidas pelos líderes do grupo principal dos nós emissores. Caso o grupo não se encontre ainda coberto, por não possuir nenhum nó encaminhador com uma rota directa, o líder de grupo envia uma mensagem para todos os membros a informar da cobertura do grupo.

Estabelecimento de rotas indirectas. Com o uso de rotas indirectas pretende-se colmatar possíveis particionamentos da rede através da utilização de um líder de grupo, aumentando-se assim o alcance dessa rota, no caso de não existir nenhuma alternativa entre nós encaminhadores. O estabelecimento de rotas indirectas têm inicio após as rotas directas terem sido já estabelecidas, com o envio de uma mensagem por parte dos nós encaminhadores que se encontrem num grupo que esteja coberto e que não possuam rotas directas para o nó sink. O esquema desencadeia-se da seguinte forma:

1. Envio de mensagem com nova rota indirecta.
2. Recepção de mensagem com rota indirecta (caso partilhe uma chave com o emissor). Envio de pedido de confirmação de rota para verificar que existe bi-direccionalidade da ligação.
3. Recepção da mensagem com pedido de confirmação. Envio da mensagem com confirmação da rota.
4. Recepção da mensagem de confirmação de rota. Envio de mensagem para líder de grupo a informar cobertura por via de uma rota indirecta.
5. Recepção da mensagem a informar cobertura. Caso o grupo ainda se encontrasse sem cobertura, reenviar essa mensagem para todos os membros do grupo.
6. Recepção da mensagem a informar cobertura, vinda do líder de grupo. Recomeçar do passo 1.

Encaminhamento seguro de dados e resistência a intrusões. A fase de encaminhamento de dados tem como objectivo o envio de uma mensagem, até ao nó *sink*, despoletada pela detecção de um evento por qualquer sensor da rede. A mensagem é enviada em *unicast* e reencaminhada até alcançar o nó *sink*. Para se introduzir critérios de resiliência pró-activa a mensagem é enviada por várias rotas redundantes (de forma aleatória e com um *fanout* que pode ser variável por parametrização). A quantidade de rotas por cada mensagem enviada depende ainda do número de nós encaminhadores presentes no grupo de filiação do nó inicial..

O processamento do encaminhamento de uma mensagem inicia-se com a detecção de um evento por parte de qualquer nó da rede, seguindo-se o respectivo envio da mensagem a ser encaminhada para o líder do seu grupo principal. Caso o nó seja ele próprio o líder de um grupo evita-se esta mensagem, passando directamente ao passo seguinte. O esquema de encaminhamento de eventos resume-se da seguinte forma:

1. Recepção (ou detecção) do evento por parte do líder de grupo. Envio da mensagem com o evento para todos os nós encaminhadores do grupo.
2. Recepção do evento, vinda do líder de grupo. Escolha do próximo nó da rota. Envio da mensagem com o evento para o nó escolhido.
3. Recepção do evento, vindo de qualquer outro nó encaminhador. Escolha do próximo nó da rota (podendo ser um líder de grupo no caso de uma rota indirecta). Envio da mensagem com o evento para o nó escolhido.
4. Repetir passo 3 até atingir o fim da rota.

Note-se que todas as escolhas de rotas podem ser feitas aleatoriamente e que existe um mecanismo de preferência por rotas directas, para minimizar o consumo de energia e diminuir a possibilidade da mensagem fluir por nós maliciosos.

Reorganização topológica. Um dos mecanismos complementares para resiliência pró-activa do protocolo baseia-se na possibilidade de reorganização dinâmica da topologia da rede (o que pode ser feito de forma probabilística ou por algum evento a pedido). Este mecanismo evita a possibilidade de estudo da topologia da rede por parte de um atacante, que poderia depois proceder a um ataque táctico a nós que estejam localizados em pontos críticos da rede por onde o tráfego tem maior probabilidade de fluir. Um nó malicioso que num determinado instante estava localizado num ponto crítico pode, após a reorganização da rede, deixar de ter um papel fulcral que poderia ser aproveitado para ataques ao encaminhamento.

Considerando a diversidade das possíveis tarefas que cada nó poderá desempenhar no protocolo bem como as repercussões de complexidade, desempenho e custo energético de reorganizações topológicas da rede, existem quatro tipos de reorganização possíveis: reorganização total da rede (equivalente à organização inicial), reorganização local de grupos, reorganização local de nós encaminhadores e reorganização por adição de nós legítimos frescos (ou activação de nós legítimos que tendo sido instalados anteriormente, se mantinham em situação de *stand-by*).

Refrescamento de chaves. O último mecanismo associado à resiliência pró-activa do sistema de encaminhamento consiste na possibilidade de refrescamento de chaves. Este mecanismo permite reduzir o impacto da possível fraqueza das chaves simétricas e tamanhos utilizáveis em redes de sensores, as quais podem sofrer ataques de força bruta ou podem ser violadas no caso de intrusão, adicionando garantias de segurança passada e segurança futura. O refrescamento frequente de chaves de forma eventualmente aleatória obrigará o atacante a executar o ataque com maior rapidez, podendo minimizar-se janela de vulnerabilidade de utilização das mesmas chaves. No sistema existem 3 tipos de refrescamentos de chaves: refrescamento de chaves de rede, de chaves de grupo e de chaves partilhadas. Note-se que as chaves para *broadcast* utilizam o mesmo esquema de renovação tal como está definido no protocolo μTESLA[7].

5 Implementação

Ambiente de simulação jProwler e modelo de comunicação por rádio 802.11.15 para MicaMotes. A pilha de segurança apresentada foi realizada e estudada tendo por base m ambiente de simulação baseado no ambiente jProwler[8], desenvolvido no ISIS (Institute for Software Integrated Systems). Este ambiente disponibiliza um motor de simulação baseado em eventos discretos, desenvolvido em JAVA e que permite prototipar, verificar e analisar protocolos de comunicação com sensores com as características do tipo de sensores usuais da família Mica Motes da Crossbow [17]. O simulador já contempla suporte de modelação para um sensor do tipo MicaMote e Mica2, actuando de forma estática ou em condições de mobilidade. Na implementação realizada são suportados dois

modelos de comunicação rádio, o modelo Gaussian e o modelo de Rayleigh, bem como uma implementação do protocolo MAC (802.15.4 ou Zigbee) para MICA2 na variante sem acknowledgment. Tal permite testar condições experimentais de simulação de comunicação com base no modelo de colisões. As condições de escalabilidade do motor de simulação permitiram criar redes de milhares de nós, sendo os tempos de referência para a formação de uma rede de 1000 cerca de 2 segundos, e de uma rede de 5000 nós da ordem de 40 segundos. Com a implementação da pilha de segurança e protocolo de encaminhamento proposto e face à adição dos novos módulos (a seguir referidos) que foram desenvolvidos no âmbito deste trabalho, a simulação foi limitada a redes de cerca de 1500 nós, sendo o seu tempo de formação na ordem dos 10 a 15 segundos.

Para a validação do protocolo apresentado foi necessária a implementação no simulador Jprowler de um módulo de simulação de consumo energético, dada a sua inexistência. O modelo de consumo energético tem por base a simulação do comportamento do TinyOS [9], onde os sensores podem estar em 3 diferentes estados: activo, *idle* ou adormecido. Todos os estados pressupõem gastos energéticos diferenciados que dependem, essencialmente, dos dispositivos do sensor modelado que se encontram ligados. Assim, no estado adormecido o sensor não tem qualquer dispositivo ligado, no estado *idle* o sensor não tem o dispositivo de comunicação por rádio ligado e no estado activo todos os dispositivos encontram-se ligados.

Foram também contemplados vários indicadores energéticos que afectam o consumo de energia dos sensores quando executam várias tarefas. Nomeadamente o envio e recepção de mensagens, cifra e decifra de dados e criação de assinaturas. Os indicadores energéticos utilizados na adição do módulo de consumo energético, foram estudados e analisados tendo sido concretizados a partir dos dados indicados em [9], [10] e [11] quer para consumo de energia associado a custo computacional, custo de comunicações e custo de operação de transições de estado e de consumo energético de primitivas criptográficas.

Implementação na pilha da base de serviços MiniSec. Para fornecer primitivas de comunicação segura aos nós (ao nível do suporte de comunicação base) implementaram-se os serviços da arquitectura MiniSec[12] como camada de comunicação segura de comunciações IEEE802.15.4 ao nível do simulador. O MiniSec é uma arquitectura que fornece segurança ao nível de comunicação MAC, fornecendo primitivas seguras para requisitos de confidencialidade, autenticação, integridade e protecção contra *message replaying*. Pormenores da implementação da pilha Minisec como camada básica de segurança das comunicações pode ser vista em [18].

6. Avaliação

Para análise e avaliação do sistema de encaminhamento proposto e implementado, foram efectuados diversos testes no ambiente de simulação anteriormente indicado, apresentando-se alguns resultados obtidos. Os testes apresentados foram obtidos para redes geradas com distribuição topológica aleatória de sensores (de acordo com o número de sensores indicado). Nas condições da simulação foram

utilizadas características de transmissão com raios de alcance de transmissão e recepção entre 20 e 30 metros, e modelação MAC para comunicações IEEE 802.15.4 em modo sem coordenador e simulação de colisões rádio. As experiências modelam uma rede cobrindo uma área quadrada plana, estimada em cerca de 4 km2.

Testes de cobertura. As condições do teste de cobertura avaliam a quantidade de nós que se encontram em grupos que têm pelo menos um nó encaminhador que tenha uma rota directa ou indirecta. Esta avaliação é essencial de modo a observarem-se os valores espectáveis de cobertura para várias parametrizações. As maiores causas de variação de cobertura no presente protocolo são relativas à quantidade de nós dispostos na rede, que irá afectar as taxas de dispersão, e o número de chaves partilhadas que cada nó possui na fase de pré-distribuição, que vai afectar a quantidade de comunicações possíveis entre nós. Os resultados são apresentados na figura 6.1 e baseiam-se em testes experimentais onde se foi aumentando a quantidade de chaves por cada nó.

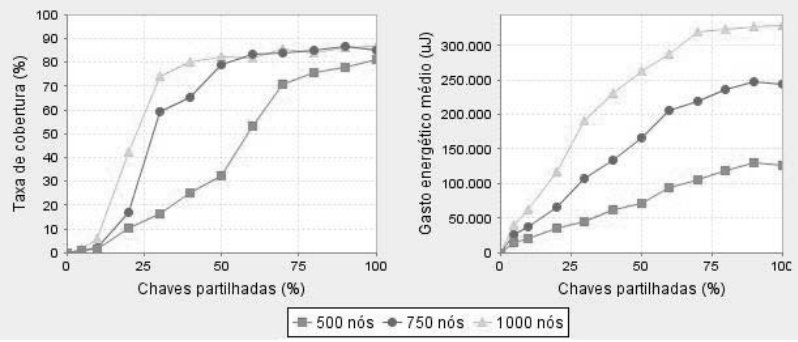


Figura 6.1 – Taxa de cobertura total (esquerda) e gasto energético médio por nó (direita) face a chaves partilhadas

Os 3 testes apresentados na figura diferem entre si no que respeita à quantidade de nós existentes nas redes geradas. A análise dos resultados obtidos permite concluir primariamente o que era expectável, podendo-se verificar que quanto mais chaves partilhadas são pré-distribuídas maior será a cobertura da rede.

É também de realçar o facto de dada uma dispersão de nós suficiente para o tamanho da rede, nomeadamente as redes de 750 e 1000 nós, os valores de cobertura tendem para um mesmo ponto, sendo que se alcançam valores aproximados mesmo sem se pré-distribuir 100% das chaves. Os resultados foram comparados com as métricas de modelos teóricos de pré-distribuição de chaves estudados para os sistemas Basic Random Key Pre-Distribution Scheme [18] e Shared-Key Thereshold Random Key Pre-Distribution Scheme [19],. Observa-se que seguem a tendência da cobertura desses sistemas, embora se revele que os modelos teóricos são muito mais optimistas que a observação experimental quando o emparelhamento de chaves tem por base um modelo de colisões

aproximado ao real. Por falta de espaço escusamo-nos de apresentar aqui essa comparação em detalhe.

Indicadores energéticos. A medição de indicadores energéticos é relevante pois permite medir o impacto energético, que as medidas de segurança induzem na rede e que poderão afectar o tempo de vida da rede. Na figura 6.1, no gráfico à direita, apresentam-se os gastos energéticos médios dos nós envolvidos durante a totalidade das fases de auto-organização da rede e selecção e estabelecimento de rotas. Os resultados apresentados baseiam-se nos 3 testes experimentais apresentados anteriormente. Estes resultados energéticos permitem ajudar na escolha das parametrizações do protocolo consoante o *trade-off* pretendido entre cobertura e energia dispendida. Na figura 6.2 apresentam-se os gastos energéticos médios dos nós envolvidos durante o encaminhamento de eventos até ao nó *sink*, em redes de 750 e 1000 nós. Todos os eventos gerados têm o tamanho de 8 bytes, não contando com os cabeçalhos do protocolo MiniSec.

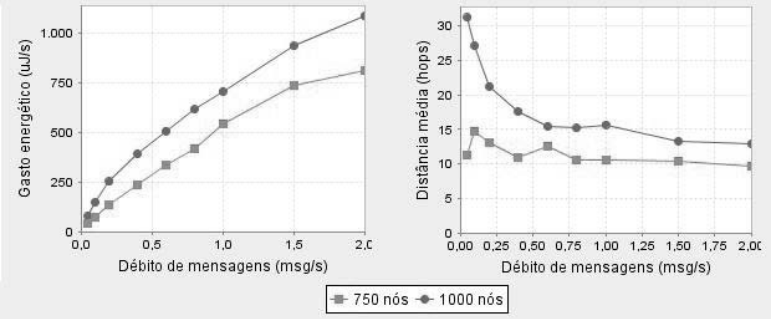


Figura 6.2 – Gasto energético médio (esquerda) e distância média percorrida pelas mensagens (direita) face a débito de mensagens

Como se estabelece como constante o tamanho de cada mensagem, irá apenas interessar observar as alterações do consumo energético face à taxa de geração de eventos pela rede. O débito de mensagens é relativo às mensagens geradas em cada segundo por um nó coberto, escolhido aleatoriamente. Adicionalmente foi também observada a distância média que cada mensagem percorre, o que indirectamente é também um indicador de consumo energético, pois como o raio de alcance dos sensores é considerado constante, cada *hop* efectuado equivale a um mesmo custo energético. Os resultados observados ao nível do gasto energético eram os esperados. As curvas tendem a perder ligeiramente a sua inclinação, dado que a distância média percorrida por mensagem tende também a diminuir com o aumento do débito das mensagens. O decréscimo da distância média percorrida, face ao aumento do débito de mensagens, é fundamentado com o decréscimo da fiabilidade. Tal pode ser observado adiante, para as mesmas circunstâncias, no caso em que se provocam menos envios do mesmo evento.

Indicadores de fiabilidade. A taxa de fiabilidade de entrega de mensagens de uma RSSF indica qual é a probabilidade de entrega de qualquer evento, sendo

interessante de observar, não só para validar o protocolo, como para estabelecer uma base comparativa em futuros testes que meçam essa mesma fiabilidade quando a rede se encontra sob ataque. Para as mesmas condições de cobertura e colisões, o parâmetro que mais influência a fiabilidade da rede é o débito de mensagens. Quantas mais mensagens forem geradas maior será a probabilidade de existirem colisões no nível físico, impedindo a sua correcta recepção. Tal como nos testes realizados para medir o consumo energético da fase de encaminhamento, a geração de mensagens é associada a nós cobertos escolhidos aleatoriamente. Foram realizados dois tipos de testes: o primeiro onde se observa a fiabilidade da entrega de mensagens em redes de 750 e 1000 nós e um segundo onde se observa a fiabilidade da entrega de mensagens, numa rede de 1000 nós, para nós que estejam a intervalos de distância variada da *sink*, considerando-se os intervalos de 1 a 15, 15 a 30 e mais que 30 *hops* de distância.

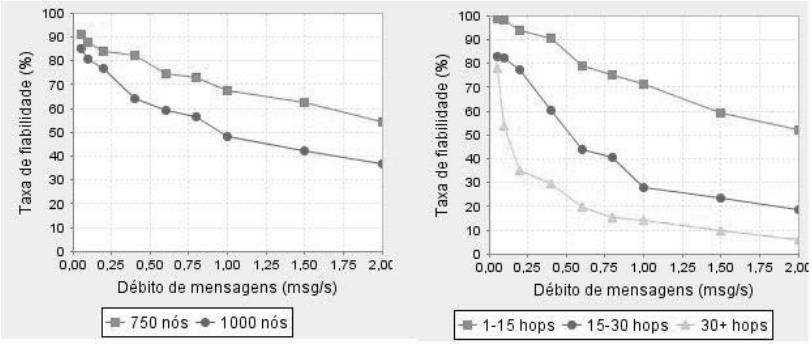


Figura 6.3 – Taxa de fiabilidade face a débito de mensagens. Variação do número de nós (esquerda) e variação da distância dos nós (direita)

Os resultados da figura 6.3 podem ser, à primeira vista, inesperados pois o mais natural seria que quantos mais nós a rede detêm, maior seriam as suas taxas de fiabilidade. Contudo é de realçar que a escolha de nós para gerarem eventos está relacionada apenas a nós cobertos, pois os restantes teriam taxa de sucesso nula, e as taxas de cobertura para uma rede de 1000 nós são superiores às de uma rede com 750 nós. Esse facto implica que uma rede de 1000 nós consegue estabelecer rotas de maior comprimento e, como se pode observar no gráfico à direita da figura 6.4, essas rotas têm uma taxa de fiabilidade inferior às restantes de menor dimensão.

Resistência face a ataques. Foram ainda realizados testes em situações em que a rede se encontra sob ataque. Os testes apresentados baseiam-se em redes de 750 e 1000 nós onde foram executados os ataques que podem causar mais estragos ao funcionamento do protocolo, nomeadamente ataques a nós encaminhadores. De forma a medir o sucesso dos mecanismos de segurança introduzidos no protocolo, no presente caso o mecanismo de multi-rotas, foram estabelecidos 2 critérios: taxa de defesa e taxa de fiabilidade. A taxa de defesa indica dos ataques que foram realizados às várias mensagens, quantos é que foram evitados por via de rotas

alternativas. A taxa de fiabilidade indica qual é a perca da taxa de entrega de mensagens face aos atacantes, que realizando os ataques impedem as mensagens de atingir o seu destino. Com a análise da figura 6.4 observa-se que há um decréscimo tanto da taxa de defesa como da taxa de fiabilidade com o aumento do número de atacantes. Isso justifica-se pois quantos mais nós encaminhadores maliciosos existirem, maior será a probabilidade de estarem posicionados nas diversas rotas alternativas, o que contribui tanto para evitar que o mecanismo multi-rotas funcione, como para evitar que as mensagens cheguem ao nó *sink*.

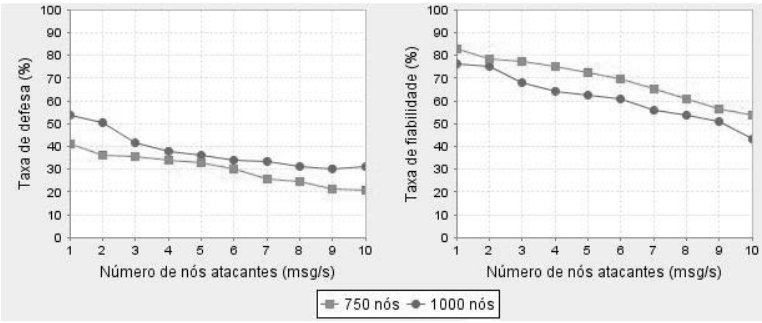


Figura 6.4 – Taxa de defesa (esquerda) e taxa de fiabilidade na entrega das mensagens (direita) face à quantidade de nós atacantes

7 Conclusões

O presente artigo propõe um sistema de encaminhamento seguro resistente a intrusões para RSSF dotado de contra-medidas face a diferentes tipos de ataques ao encaminhamento. Os serviços de segurança são introduzidos a diferentes níveis de uma pilha que dá suporte a um protocolo de encaminhamento seguro que utiliza mecanismos probabilísticos e critérios de resiliência activa para minimizar o impacto de ataques que podem ser desencadeados contra as comunicações IEEE801.15.4 e podem abarcar a intrusão física e comprometimento de sensores da rede. O artigo apresenta os aspectos de implementação e avaliação experimental do sistema numa base de simulação para RSSF de milhares de nós. Apresentam-se vários testes que permitem concluir que é possível obter bons indicadores de cobertura, fiabilidade e resistência a ataques, de acordo com uma correcta parametrização do sistema.

O trabalho em curso e os desenvolvimentos futuros têm em vista uma validação extensiva do protocolo e dos seus mecanismos de segurança, bem como a extensão do estudo da verificação do impacto dos diversos mecanismos de segurança em termos de consumos energéticos. São igualmente relevantes os resultados de outros testes já efectuados e em curso, em que se visa medir o impacto na cobertura e fiabilidade, motivada por reorganizações topológicas aleatórias e refrescamentos dinâmico e aleatório de chaves, desencadeados durante a operação normal do encaminhamento de informação na rede

Foram estudadas e comparadas as características do sistema proposto com outros protocolos de encaminhamento seguro em RSSF. Por limitações de espaço apenas se apresenta a visão comparativa face aos protocolos SecLEACH[13] e Clean-Slate[14], cujas características se aproximam do protocolo apresentado.

O SecLEACH é um protocolo de encaminhamento seguro orientado a grupos, que adiciona condições de segurança a um protocolo de encaminhamento para RSSF anteriormente existente (o protocolo LEACH). Para o efeito são introduzidos mecanismos criptográficos baseados em criptografia simétrica e chaves secretas pré-distribuidas aleatoriamente para garantir confidencialidade e autenticação. O protocolo inclui um mecanismo de renovação topológica da rede, visto pelos autores como um mecanismo de manutenção energética, mas que também permite introduzir critérios de resistência a ataques que podem basear-se no estudo da topologia da rede. Contrariamente ao protocolo proposto, o SecLEACH não considera defesas contra intrusões físicas ao sensores..

O Clean-Slate é um protocolo de encaminhamento seguro hierárquico, orientado a grupos, tendo sido criado de raiz com vista a cumprir condições de segurança. Possui mecanismos criptográficos inspirados em criptografia simétrica e sínteses seguras de mensagens para obter requisitos de confidencialidade, autenticação, integridade e protecção contra ataques de *replaying*. Adicionalmente introduz mecanismos de segurança pensados para protecção face a eventuais intrusões físicas, nomeadamente, mecanismos de encaminhamento multi-rota, de renovação topológica e de detecção de intrusões. Face ao protocolo proposto, o Clean-Slate não comporta mecanismos de refrescamento de chaves e usa uma parametrização mais rígida do mecanismo de encaminhamento multi-rota, que não permite parametrizar o número de caminhos redundantes. O mecanismo de renovação topológica está apenas pensado para manutenção energética. Por outro lado não são contemplados nem estudados formas de adição dinâmica de nós nem de reconfigurações topológicas da rede ou o seu eventual impacto no funcionamento do protocolo.

Referências

1. IEEE 802.15.4 Standard, Wireless MAC and PHY Specifications for Low Rate Wireless Personal Area Networks (WPANs), IEEE Standard, 802.15 IEEE Group, 2006
2. D. Dolev, and A. C. Yao, "On the security of public key protocols", Information Theory, IEEE Transactions on Volume 29, Issue 2, Mar 1983 Page(s):198 - 208.
3. C. Karlof, and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures." Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on
4. Y.-C. Wang, and Y.-C. Tseng, "Attacks and defenses of routing mechanisms in ad hoc and sensor networks," Security in Sensor Networks, Y. Xiao, ed.: Auerbach Publications, 2006.
5. P. Sousa, N. F. Neves, P. Veríssimo et al., "Proactive resilience through architectural hybridization," in Proceedings of the 2006 ACM symposium on Applied computing, Dijon, France, 2006.

6. Haowen Chan, A. Perrig, and D. Song. "Random key predistribution schemes for sensor networks" in Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California, May 2003.
7. A. Perrig, R. Szewczyk, J. D. Tygar et al., "SPINS: security protocols for sensor networks," Wireless Networks, vol. 8, no. 5, pp. 521-534, 2002.
8. "jProwler," http://w3.isis.vanderbilt.edu/projects/nest/jprowler/index.html.
9. Manoussos Athanassoulis, Ioannis Alagiannis and Stathes Hadjiefthymiades. "Energy Efficiency in Wireless Sensor Networks: A Utility-Based Architecture" in European Wireless 2007 (EW 2007), Paris, France, April 1-4, 2007
10. G. Guimaraes, E. Souto, D. Sadok et tal., "Evaluation of Security Mechanisms in Wireless Sensor Networks" in ICW '05: Proceedings of the 2005 Systems Communications
11. A. S. Wander, N. Gura, H. Eberle et tal "Energy analysis of public-key cryptography for wireless sensor networks" in Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference.
12. M. Luk, G. Mezzour, A. Perrig et al., "MiniSec: a secure sensor network communication architecture," in Proceedings of the 6th international conference on Information processing in sensor networks, Cambridge, Massachusetts, USA, 2007.
13. L. B. Oliveira, H. C. Wong, M. Bern et al., "SecLEACH - A Random Key Distribution Solution for Securing Clustered Sensor Networks." pp. 145-154.
14. B. Parno, M. Luk, E. Gaustad et al., "Secure sensor network routing: a clean-slate approach," in Proceedings of the 2006 ACM CoNEXT conference, Lisboa, Portugal, 2006.
15. J. A. Gutiérrez, E. H. Callaway, Jr., and R. Barret, "Low-Rate Wireless Personal Area Networks", IEEE Press, New York, 2004
16. V. Misic, J. Fung, J. Misic, "MAC Layer Attacks in 802.15.4 Sensor Networks", in Security in Sensor Networks, Ed. By Yang Xiao, pp. 27-44, Auerbach Publications, 2006
17. http://www.xbow.com/Products/wproductsoverview.aspx
18. Pedro Amaral, Henrique Domingos, Uma Arquitectura de Segurança para um Serviço de Encaminhamento para Redes de Sensores sem Fios, TECHNICAL REPORT, DI-FCT-UNL, Junho 2009

Sessão 2A: Compiladores, Linguagens de Programação e Tecnologias Relacionadas (CORTA)

VisualLISA: A Domain Specific Visual Language for Attribute Grammars

Nuno Oliveira[1], Maria João Varanda Pereira[2], Pedro Rangel Henriques[1], Daniela da Cruz[1], and Bastian Cramer[3]

[1] University of Minho - Department of Computer Science,
Campus de Gualtar, 4715-057, Braga, Portugal
{nunooliveira,prh,danieladacruz}@di.uminho.pt
[2] Polytechnic Institute of Bragança
Campus de Sta. Apolónia, Apartado 134 - 5301-857, Bragança, Portugal
mjoao@ipb.pt
[3] University of Paderborn - Department of Informatics
Fürstenallee 11, 33102, Paderborn, Germany
bcramer@upb.de

Abstract. The focus of this paper is on the translation of AG formalisms into a new visual language, and on the development of the associated programming environment. We present a solution for rapid development of VisualLISA editor using DEViL. This tool uses traditional attribute grammars, to specify the language's syntax and semantics, extended by visual representations to be associated with grammar symbols. From these specifications a visual programming environment is automatically generated. This environment allows us to edit a visual description of an AG that is automatically translated into textual notations. In this paper, we emphasize the design and implementation of VisualLISA that is intended to be helpful for beginners and rapid development of small AGs.

1 Introduction

Attribute Grammars (AGs) [1] introduced by Knuth, 40 years ago, is a powerful and well-known formalism used to create language processors. An AG can be formally defined as the following tuple: $AG = (G, A, R, C)$, where G is a context-free grammar, A is the set of attributes, R is the set of evaluation rules, and C is the set of contextual conditions. Each attribute has a type, and represents a specific property of a symbol X; we write $X.a$ to indicate that attribute a is an element of the set of attributes of X, denoted by $A(X)$. For each X (terminal or non-terminal), $A(X)$ is divided into two disjoint sets: the *inherited* and the *synthesized* attributes. Each R is a set of formulas, like $X.a = func(..., Y.b, ...)$, that define how to compute, in the precise context of a production, the value of each attribute. Each C is a set of predicates, $pred(..., X.a, ...)$, describing the requirements that must be satisfied in the precise context of a production.

As can be deduced from this complex definition of AGs they are not as easy to specify as people would desire because there is a gap between the problem solution (the desired output) and the source language that must be interpreted. The user must take

care on choosing the appropriate attributes and their evaluation rules. Since the beginning, the literature related with compilers presents AGs using syntax trees decorated with attributes. So it is usual to sketch up on paper trees with attributes representing an AG. This strategy allows the developers to imagine a global solution of the problem (in a higher abstraction level) and to detect complex dependencies between attributes, symbols and functions, avoiding spending time with syntax details. However, such informal drawings require the designer to translate them manually into the input notation of a compiler generator. The person who drew it must go through the translation of the pencil strokes into the concrete syntax of the compiler generator. These inconveniences make the developers avoid the usage of AGs and go through non systematic ways to implement the languages and supporting tools. So, in this paper, we develop a *Visual Language* (VL), as a meta-language to write AGs, based on a previous conceptualization that we have proposed in [2].

VLs and consequently the *Visual Programming Languages* (VPLs) aim at offering the possibility to solve complex problems by describing their properties or their behavior through graphical/iconic definitions [3]. Icons are used to be composed in a space with two or more dimensions, defining sentences that are formally accepted by parsers, where shape, color and relative position of the icons are relevant issues. A visual programming language implies the existence of a *Visual Programming Environment* (VPE) [4, 5], because its absence makes the language useless. Commonly, a visual programming environment consists of an editor, enriched by several tools to analyze, to process and to transform the drawings.

LISA [6, 7] is a compiler generator based on attribute grammars, developed at University of Maribor at Slovenia. It generates a compiler and several other graphical tools from a textual AG specification, as can be seen in [8]. Since it generates a set of useful visual/graphical tools, it would be desirable to have a graphical way of specifying the AG too.

So the main idea is to enhance the front-end of LISA by developing a VPE, named VisualLISA, that assures the possibility to specify AGs visually, and to translate them into LISA specifications or, alternatively, into a universal XML representation designed to support generic AG specifications. The main objective of this environment is to diminish the difficulties regarding the specification of AGs not only in LISA but also for other similar systems.

The visual programming environment was automatically generated by DEViL, our choice among many other tools studied; so in this paper, the system is introduced and its use explained. However, our objective in this paper is not concerned with the discussion of compiler development tools, but show the befits of using an effective one.

In Section 2, VisualLISA language and editor are informally described. In Section 3 the language is formally specified, defining syntactic rules, semantic constraints and translation scheme, which requires the description of the target language — in our case we present LISA's syntax (the first target), and $\mathcal{X}$AGra, an XML notation for AGs that we designed specially to support a universal representation." In Section 4, the DEViL generator framework, used for the automatic generation of the visual editor, will be presented. In Section 5, following the informal conception and its formalization, using DEViL, the visual language and the editor implementation is shown. An overview

on how to use the editor to describe an AG, is given in Section 6 before concluding the paper in Section 7.

2 VisualLISA - A Domain Specific Visual Language

For many years we have been thinking and working with AGs. Inevitably we created an abstract mental representation of how it can be regarded and then sketched, for an easier comprehension and use. So we decided to implement a framework that follows that representation. The conception of that framework is described in this section.

2.1 The Language Conception

VisualLISA, as a new *Domain Specific Visual Language* (DSVL) for attribute grammar specification, shall have an attractive and comprehensible layout, besides the easiness of specifying the grammar model.

We think that a desirable way to draw an AG is to make it production oriented, and from there design each production as a tree. The *Right-Hand Side* (RHS) symbols should be connected, by means of a visible line, to the *Left-Hand Side* (LHS) symbol. The attributes should be connected to the respective symbols, using a connection line different from the one referred before, as both have different purposes (see Figure 1). The rules to compute the values of each attribute should exhibit the shape of a function with its arguments (input attributes) and its results (the output attributes). Two kinds of functions should be represented: the identity function (when we just want to copy values) or a generic function (for other kind of computations). Often a production has a considerable number of attributes and nontrivial computations. Therefore we think that for visualization purposes, the layout of each production should work as a reusable template to draw several computation rules. Hence, the rules are drawn separated from each other, but associated to a production.

Figure 1 depicts a simple example of the conception we made for the language. This example is based on a simple AG , called Students Grammar, used to process a list of students, described by their names and ages. The objective of the attribute grammar is to sum the ages of all the students. This grammar can be textually defined as shown in Listing 1.1. In Figure 1 just production P1 is shown. The attributes are associated to the symbols of the production. Moreover, the production has a semantic rule that computes the value of the LHS's attribute, *sum*, by adding the value of the attributes in the RHS symbols, *sum* and *age*.

Listing 1.1. Students Grammar

```
1 P1: Students → Student Students
2         {Students0.sum = Student.age + Students1.sum}
3 P2: Students → Student
4         {Students.sum = Student.age}
5 P3: Student → name age
6         {Student.age = age.value}
```

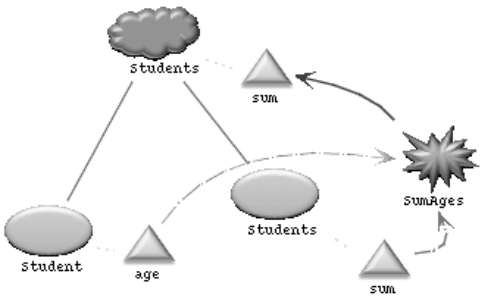


Fig. 1. VisualLISA Conception - Production P1.

2.2 The Editor's Main Features

VisualLISA editor should be compliant with the idea of offering a nice and non error-prone way of sketching the AG, as a first step; and an easy translation of the model into a target language, as a second step. So, three main features are highlighted: (i) syntax validation, (ii) semantics verification and (iii) code generation. The syntax validation restricts some spatial combinations among the icons of the language. In order to avoid syntactic mistakes, the edition should be syntax-directed. The semantics verification copes with the static and dynamic semantics of the language. Finally, the code generation feature generates code from the drawings sketched up. The target code would be LISAsl or $\mathcal{X}$AGra. LISAsl specification generated is intended to be passed to LISA system in a straightforward step. $\mathcal{X}$AGra specification generated is intended to give the system more versatility and further usage perspectives.

3 Specification of VisualLISA

The specification of VisualLISA bases on three main issues: $i)$ the definition of the underlying language's syntax; $ii)$ the language semantics and $iii)$ the description of the textual specifications into which the iconic compositions will be translated.

3.1 Syntax

The *Picture Layout Grammar* (PLG) formalism [9], is an attribute grammar to formally specify visual languages. It assumes the existence of pre-defined terminal symbols and a set of spatial relation operators. Our acquaintance with PLG formalism, from previous works, led us to use it to specify the syntax of VisualLISA. Listings 1.2 present some parts of the language specification.

Figure 2 shows the concrete and connector icons used for VisualLISA specifications. *LeftSymbol* is the LHS of a production, while *NonTerminal* and *Terminal* are used to compose the RHS. The second line of icons in Figure 2 presents the several classes of attributes. *Function* and *Identity*, both representing operations, are used to compute

the attribute values. The other icons connect the concrete symbols with each other, to rig up the AG.

Listing 1.2. VisualLISA Partial Syntax Definition.

```
AG → contains(VIEW, ROOT)

VIEW → labels(text, rectangle)

ROOT → left_to(PRODS, SPECS)

SPECS → contains(VIEW,
          over(LEXEMES, USER_FUNCS))

PRODS → group_of(SEMPROD)

SEMPROD → contains(VIEW, left_to(
   group_of(group_of(RULE_ELEM)),
   group_of(AG_ELEM)))

AG_ELEM → LEFT_SYMBOL
        | NON_TERMINAL
        | TERMINAL
        | SYNT_ATTRIBUTE
        | INH_ATTRIBUTE
        | TREE_BRANCH
        | INT_ATTRIBUTE
        | SYNT_CONNECTION
        | INH_CONNECTION
        | INT_CONNECTION
```

```
RULE_ELEM → FUNCTION
          | IDENTITY
          | FUNCTION_ARG
          | FUNCTION_OUT

TERMINAL → labels(text, rectangle)

INT_ATTRIBUTE → labels(text, triangle)

INT_CONNECTION → points_from(
                   points_to(
                     dash_line,
                     ∼INT_ATTRIBUTE),
                     ∼TERMINAL)

FUNCTION → over(rectangle, text)

FUNCTION_OUT → points_from(
             points_to(arrow,
             ∼INH_ATTRIBUTE),
             ∼FUNCTION)
               | points_from(
             points_to(arrow,
             ∼SYNT_ATTRIBUTE),
             ∼FUNCTION)
```

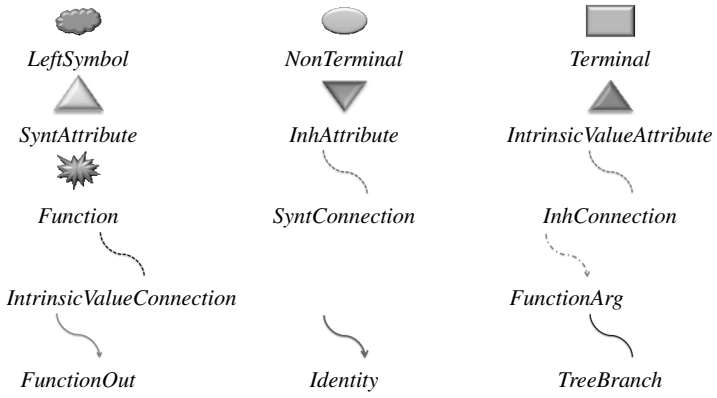


Fig. 2. The Icons of VisualLISA

3.2 Semantics

In order to correctly specify an AG, many semantic constraints must hold. These constraints are related with the attribute values that depend on the context in which the associated symbols occur in a sentence. We separated these constraints into two major groups. One concerning the syntactic rules, *Production Constraints* (PC), and another the respective computation rules, *Computation Rules Constraints* (CRC).

The constraints concerning the VisualLISA's semantic correctness, are not listed in this document, for the sake of the space. The complete set of constraints can be seen in [10].

3.3 Translation

The translation ($\mathcal{L}_s \rightarrow \tau \rightarrow \mathcal{L}_t$) is the transformation of a source language into a target language. τ is a mapping between the productions of the $\mathcal{L}_s$ (VisualLISA) and the fragments of $\mathcal{L}_t$ (LISAsl $\cup$ $\mathcal{X}$AGra). These fragments will be specified in this sub-section.

A *Context Free Grammar* (CFG) is a formal and robust way of representing LISA specifications' structure. Listing 1.3 presents that high-level CFG.

Listing 1.3. LISA structure in a CFG.

```
1  p1: LisaML          → language id { Body }
2  p2: Body            → Lexicon Attributes Productions Methods
3  p3: Lexicon         → lexicon { LexBody }
4  p4: LexBody         → (regName regExp)*
5  p5: Attributes      → attributes (type symbol . attName ;)*
6  p6: Productions     → rule id { Derivation } ;
7  p7: Derivation      → symbol ::= Symbs compute { SemOperations }
8  p8: Symbs           → symbol+
9  p9:                  | epsilon
10 p10: SemOperations  → symbol . attName = Operation ;
11 p11: Operation      → ...
12 p12: Methods        → method id { javaDeclarations }
```

Reserved words, written in bold, indicate, in its majority, the beginning of important fragments. The fact of separating the structure in smaller chunks, makes the process of generating code easier and modular.

Regarding the literature, there is not an XML standard notation for AGs. So that, $\mathcal{X}$AGra was defined using a schema. The whole structure of this schema can be separated into five big fragments: i) *symbols* — where the terminal, nonterminal and the start symbols are defined; ii) *attributesDecl* — where information about the attributes and the symbols to which they are associated is stored; iii) *semanticProds* — where the productions and the semantic rules are declared: in each production, the LHS, the RHS and the attribute computations are defined in a very modular way; iv) *importations* — where the modules or packages necessary to perform the computations are declared and v) *functions* — is the element where the user declare necessary functions. A more detailed explanation about these elements, their sub-elements and attributes can be seen in [10].

4 DEViL - A Tool for Automatic Generation of Visual Programming Environments

We searched for VPE generators like MetaEdit+ [11], but their commercial nature was not viable for an academic research. Also, we experimented VLDesk [12], Tiger [13], Atom^3 [14] and other similar tools, however none of them gave us the flexibility that DEViL offered, as described below.

DEViL system generates editors for visual languages from high-level specifications. DEViL (*Development Environment for Visual Languages*) has been developed at

the University of Paderborn in Germany and is used in many nameable industrial and educational projects.

The editors generated by DEViL offer syntax-directed editing and all features of commonly used editors like multi-document environment, copy-and-paste, printing, save and load of examples. Usability of the generated editors and DEViL and can be found in [15]. DEViL is based on the compiler generator framework Eli [16], hence all of Eli's features can be used as well. Specially the semantic analysis module can be used to verify a visual language instance and to produce a source-to-source translation.

To specify an editor in DEViL we have to define the semantic model of the visual language at first. It is defined by the domain specific language *DEViL Structure Specification Language* (DSSL) which is inspired by object-oriented languages and offers classes, inheritance, aggregation and the definition of attributes. The next specification step is to define a concrete graphical representation for the visual language. It is done by attaching so called visual patterns to the semantic model of the VL specified in DSSL. Classes and attributes of DSSL inherit from these visual patterns. Visual patterns [17] describe in what way parts of the syntax tree of the VL are represented graphically, e.g. we can model that some part should be represented as a "set" or as a "matrix". DEViL offers a huge library of precoined patterns like formulae, lists, tables or image primitives. All visual patterns can be adapted through control attributes. E.g. we can define paddings or colors of all graphical primitives. Technically visual patterns are decorated to the syntax tree by specifying some easy inheritance rules in a DSL called LIDO.

To analyse the visual language, DEViL offers several ways. The first one results from the fact that editors generated by DEViL are syntax directed. Hence, the user cannot construct *wrong* instances of the VL It is limited by its syntax and cardinalities expressed in DSSL. Another way is to define check rules e.g. to check the range of an integer attribute or to do a simple name analysis on a name attribute. To navigate through the structure tree of the VL, DEViL offers so called path expressions which are inspired by XPath. They can be used in a small simple DSL to reach every node in the tree. After analysis, DEViL can generate code from the VL instance. This is done with the help of Eli which offers unparsers, template mechanism (*Pattern-based Text Generator* — PTG) and the well-known attribute evaluators from compiler construction.

5 Implementation of VisualLISA

After having all the requirements formally specified (Section 3) and a VPE generator chosen, the implementation of VisualLISA is a straightforward work and can be systematized in four main steps: i) Abstract Syntax Specification; ii) Interaction and Layout Definition; iii) Semantics Implementation; and iv) Code Generation.

5.1 Abstract Syntax

The specification of the abstract syntax of VisualLISA, in DEViL, follows an object-oriented notation, as referred previously. This means that the nonterminal symbols of the grammar are defined modularly: the symbols can be seen as classes and the attributes of the symbols as class attributes.

The syntax of the visual language is determined by the relations among their symbols. Therefore, for an high level representation of the language's syntax, a class diagram can be used. This diagram should meet the structure of the PLG model in Figure 1.2. The final specification for the language is then an easy manual process of converting the diagram into DSSL. Figure 3 shows a small example of the diagram and the resultant specification.

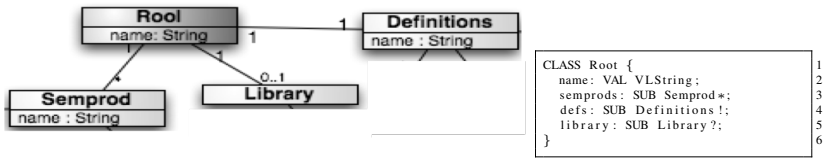


Fig. 3. Class Diagram and Respective DEViL Notation

There are two types of classes in this notation: concrete and abstract. The concrete classes are used to produce a syntax tree, which is manipulated in the other steps of the environment implementation. The abstract classes are used to group concrete classes with the purpose of defining syntactic constraints. These classes generate the syntax-directed editor.

In order to make possible the specification of separated computation rules reusing the same layout of a production, we used DEViL's concept of coupled structures [18]. It couples the syntactic structure of two structure tree — for VisualLISA we used the structure of symbol *Semprod*, which is used to model a production. In practice, it means that the layout defined for a production is replicated whenever a computation rule is defined, maintaining both models synchronized all the time.

5.2 Interaction and Layout

The implementation of this part, in DEViL, consists in the definition of views. A view can be seen as a window with a dock and an editing area where the language icons are used to specify the drawing.

VisualLISA Editor is based in four views: *rootView*, to create a list of productions; *prodsView*, to model the production layout; *rulesView*, to specify the semantic rules reusing the production layout and *defsView*, to declare global definitions of the grammar.

At first the buttons of the dock, used to drag structure-objects into the edition area, are defined. Then the visual shape of the symbols of the grammar for the respective view are defined. Figure 4 shows parts of view definitions and the respective results in the editor. The code on the left side of Figure 4 defines the view, the buttons and the behavior of the buttons. The default action is the insertion of a symbol in the editing

area, but it can be extended. The bluish rectangular image represents the button resultant from that code.

```
1 VIEW rootView ROOT Root{
2   BUTTON IMAGE "img::btnProd"
3     INSERTS Semprod
4       INFO "Production";
5 }
```


```
SYMBOL pview_NonTerminal                1
INHERITS VPForm                         2
COMPUTE                                 3
  SYNT.drawing =                        4
      ADDROF(ntDrawing);                5
END;                                    6
```

Fig. 4. Parts of View Definitions and Respective Visual Outcomes

Symbol *NonTerminal* is represented by the orange oval in Figure 4. The code on the right reveals the semantic computation to define the shape of that symbol. Shape and other visual aspects of the tree-grammar symbols are automatically defined associating, by inheritance, visual patterns.

5.3 Semantics

As long as `VisualLISA` is defined by an `AG`, the contextual conditions could be checked using the traditional approach. `DEViL` is very flexible and offers some other ways to implement this verification module. The approach used to develop `VisualLISA`, is completely focused on the contexts of the generated syntax tree. `DEViL` offers a tree-walker, that traverses the tree and for a given context — a symbol of that tree — executes a verification code, returning an error whenever it occurs. With this approach it is easy to define data-structures helping the verification process. This approach is very similar to the generic `AG` approach, but instead of attributes and semantic rules, it uses variables which are assigned by the result of queries on the tree of the model.

Listing 1.4 shows the code for the implementation of a constraint defined in [10].

Listing 1.4. Implementation of Constraint: "Every *NonTerminal* specified in the grammar must be root of one production"

```
1  checkutil::addCheck Semprod {
2   set n [llength [c::getList {$obj.grammarElements.CHILDREN[LeftSymbol]}]]
3   set symbName [c::get {$obj.name.VALUE}]
4   if { $n == 0 } {
5    return "Production '$symbName' must have one Root symbol!"
6   } elseif {$n > 1} {
7    return "Production '$symbName' must have only one Root symbol!"
8   }
9   return ""
10 }
```

A considerable amount of the constraints defined in Section 3.2 were verified resorting to the Identifier Table, which is a well known strategy in language processing for that purpose.

5.4 Code Generation

The last step of the implementation, concerning the translation of the visual `AG` into `LISA` or $\mathcal{X}$`AGra`, can be done using the `AG` underlying the visual language (as usual

in language processing). For this task, DEViL supports i) powerful mechanisms to ease the semantic rules definition; ii) facilities for extend the semantic rules by using functions and iii) template language (PTG of Eli system) incorporation to structure out the output code.

The use of patterns (templates) is not mandatory. But, as seen in the formal definition of LISA and $\mathcal{X}$AGra notation (Section 3.3), both of them have static parts which do not vary from specification to specification. Hence templates are very handy here. Even with templates, the translation of the visual AG into text is not an easy task. Some problems arise from the fact that there is not a notion of order in a visual specification. We used auxiliary functions to sort the RHS symbols by regarding their disposition over an imaginary X-axe. Based on this approach we also solved issues like the numbering of repeated symbols in the production definition.

The templates (invoked like functions) and the auxiliary functions, together with LIDO-specific entities like INCLUDING or CONSTITUENTS, were assembled into semantic rules. These entities collect the values of attributes from different productions at the same time. Then these values are used by templates or functions in order to define the translation module. One module was defined for each target notation. New translation modules can be added, for supporting new target notations.

6 AG Specifcation in VisualLISA

Figure 5 shows the editor appearance, presenting the four views of our editor.

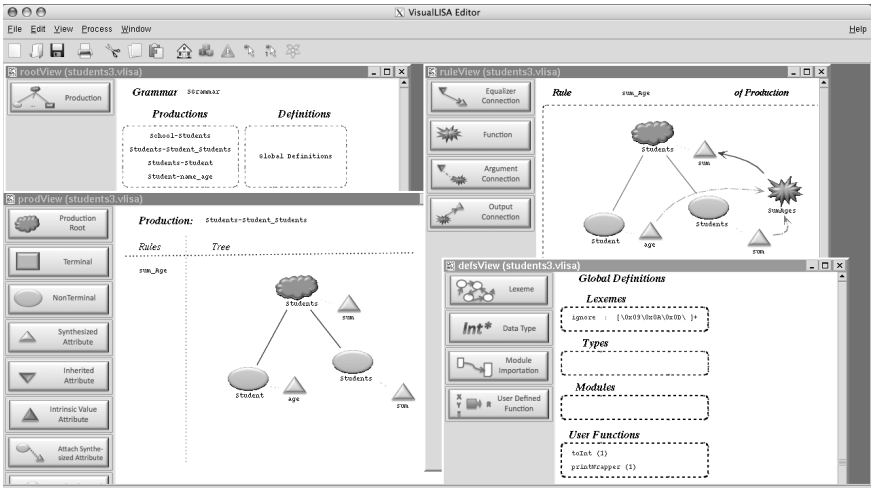


Fig. 5. VisualLISA Editor Environment

To specify an attribute grammar the user starts by declaring the productions (in *rootView*) and rigging them up by dragging the symbols from the dock to the editing area (in *prodsView*), as commonly done in VPEs. The combination of the symbols is almost automatic, since the editing is syntax-directed. When the production is specified, and the attributes are already attached to the symbols, the next step is to define the computation rules. Once again, the user drags the symbols from the dock, in *rulesView*, to the editing area, and compounds the computations by linking attributes to each other using functions. Sometimes it is necessary to resort to user-defined functions that should be described in *defsView*. In addition, he can import packages, define new data-types or define global lexemes.

The AG example addressed in Section 2.1 is composed of three productions with associated semantic rules. Production P1 is already shown in Figure 1. The other two productions are shown in Figure 6.

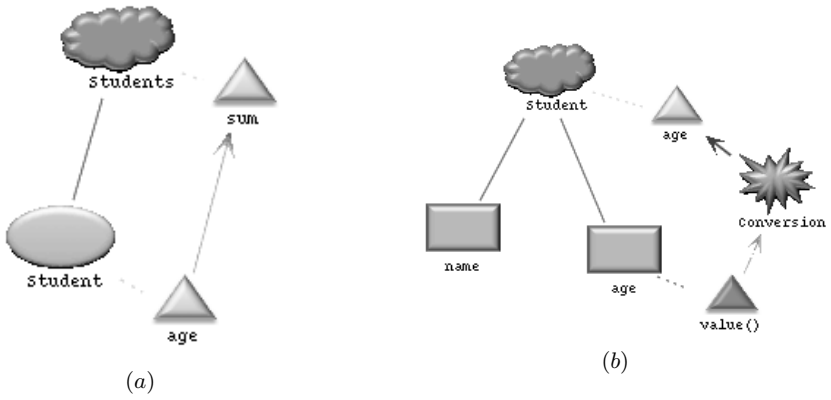


Fig. 6. Specification of Productions P2 and P3, (a) and (b) respectively, with associated semantic rules.

In production P2, the identity function is used to copy the value of the attribute *age* to the attribute *sum*. Production P3 makes use of terminal symbols and associated intrinsic values. The computation rule, in this production, is based on the conversion of the textual value of the age into an integer.

When the grammar is completely specified and semantically correct, code can be generated. Listing 1.5 shows, in LISA and $\mathcal{X}$AGra notations, the code generated for production P1 in Figure 1.

Listing 1.5. Code Generated for $a)$ LISA and $b)$ $\mathcal{X}$AGra

```
1
2 a)
3 rule Student_1 {
4   STUDENTS ::= STUDENT STUDENTS compute {
5     STUDENTS.sum = STUDENTS[1].sum + STUDENT.age;
6   };
```

```
}


b)
<semanticProd name="Students_1">
 <lhs nt="Students" />
 <rhs>
  <element symbol="Student" />
  <element symbol="Students" />
 </rhs>
 <computations>
  <computation name="sumAges">
   <assignedAttribute att="Students.sum" position="0" />
   <operation returnType="int">
    <argument att="Students.sum" position="2" />
    <arguemnt att="Student.age" position="1" />
    <modus> 1+2 </modus>
   </operation>
  </computation>
 </computations>
</semanticProd>
```

7 Conclusion

After many years working in specification and implementation of compilers supported by *Attribute Grammars*, it became clear that a modular and reusable approach to AG development is highly recommendable and necessary. On the other hand, the work on program comprehension tools emphasized the importance of software/data visualization. The combination of those two areas of R&D with a third one, the development of *Visual Languages*, gave rise to the proposal of creating a VL for AGs. The obligation to write text-based AG specifications imposed by several compiler generator tools and the habitual way of sketching AGs on paper in the form of a decorated tree, shortening the gap to the mental representation of an AG, reinforced the appropriateness of that proposal.

In this paper we introduced VisualLISA, a new *Domain Specific Visual Language* (DSVL), which enables the specification of AGs in a visual manner and the translation of that visual AG into LISA or $\mathcal{X}$AGra (an XML notation to support generic AG specifications). We were mainly concerned with the design of the language and its formal and automatic implementation. In this phase of our project we neither focused on the usability of the language nor on its scalability. We focused on the specification, aiming at showing the formal work behind the visual outcome, and on the implementation of the underlying environment to specify AGs. At this point we highlighted the use of DEViL in order to create the desired environment, through a systematic approach of development. Also, an example was presented to show the steps to build an AG with VisualLISA.

In the future, it is our objective to perform at least, two experimental studies involving VisualLISA: one to assess the usability of the language regarding the visual vs textual approaches for developing AGs; and another one to test the scalability of the language and environment, regarding the hypothesis that it was created to cope with small AGs. We are also interested in assessing the comprehension of AGs; maybe VisualLISA would be very handy on this matter, working as AGs visualizer.

References

1. Knuth, D.E.: Semantics of context-free languages. Theory of Computing Systems **2**(2) (June 1968) 127–145
2. Pereira, M.J.V., Mernik, M., da Cruz, D., Henriques, P.R.: VisualLISA: a visual interface for an attribute grammar based compiler-compiler (short paper). In: CoRTA08 — Compilers, Related Technologies and Applications, Bragança, Portugal. (July 2008)
3. Boshernitsan, M., Downes, M.: Visual programming languages: A survey. Technical report, University of California, Berkeley, California 94720 (December 2004)
4. Kastens, U., Schmidt, C.: VL-Eli: A generator for visual languages - system demonstration. Electr. Notes Theor. Comput. Sci. **65**(3) (2002)
5. Costagliola, G., Tortora, G., Orefice, S., De Lucia, A.: Automatic generation of visual programming environments. Computer **28**(3) (1995) 56–66
6. Mernik, M., Lenič, M., Avdičaušević, E., Žumer, V.: LISA: An interactive environment for programming language development. Compiler Construction (2002) 1–4
7. Mernik, M., Korbar, N., Žumer, V.: LISA: a tool for automatic language implementation. SIGPLAN Not. **30**(4) (1995) 71–79
8. Henriques, P.R., Pereira, M.J.V., Mernik, M., Lenič, M., Gray, J., Wu, H.: Automatic generation of language-based tools using the lisa system. Software, IEE Proceedings - **152**(2) (2005) 54–69
9. Golin, E.J.: A Method for the Specification and Parsing of Visual Languages. PhD thesis, Brown University, Department of Computer Science, Providence, RI, USA (May 1991)
10. Oliveira, N., Pereira, M.J.V., da Cruz, D., Henriques, P.R.: VisualLISA. Technical report, Universidade do Minho (February 2009) `www.di.uminho.pt/~gepl/VisualLISA/documentation.php`.
11. Tolvanen, J.P., Rossi, M.: Metaedit+: defining and using domain-specific modeling languages and code generators. In: OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, New York, NY, USA, ACM (2003) 92–93
12. Costagliola, G., Deufemia, V., Polese, G.: A framework for modeling and implementing visual notations with applications to software engineering. ACM Trans. Softw. Eng. Methodol. **13**(4) (2004) 431–487
13. Ehrig, K., Ermel, C., Hänsgen, S., Taentzer, G.: Generation of visual editors as eclipse plug-ins. In: ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, New York, NY, USA, ACM Press (2005) 134–143
14. de Lara, J., Vangheluwe, H.: Atom3: A tool for multi-formalism and meta-modelling. In: FASE '02: Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering, London, UK, Springer-Verlag (2002) 174–188
15. 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, Coeur d'Alène, Idaho, USA, IEEE Computer Society Press (September 2007) 231–238
16. Gray, R.W., Heuring, V.P., Levi, S.P., Sloane, A.M., Waite, W.M.: Eli: A complete, flexible compiler construction system. Communications of the ACM **35**(2) (February 1992) 121–131
17. 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 (July 2006)
18. Schmidt, C.: Generierung von Struktureditoren für anspruchsvolle visuelle Sprachen. Dissertation, Universität Paderborn (January 2006)

ATOM: Automatic Transaction-Oriented Memoization*

Hugo Rito and João Cachopo

INESC-ID Lisboa/Technical University of Lisbon
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
hugo.rito@ist.utl.pt, joao.cachopo@ist.utl.pt

Abstract. Memoization is a well-known technique for improving the performance of a program, but it has been confined mostly to functional programming, because it applies only to pure functions.
In this paper, we propose an extended memoization approach that takes advantage of the support provided by a Software Transactional Memory (STM) to remove many of the limitations of traditional memoization. We argue that our extended memoization system is the first to suit the needs of object-oriented programming, and show that it may be implemented almost for free in systems that use an STM.
We describe the major design and implementation decisions of our system and present results for a benchmark that show a 3-fold increase in the throughput of the system when using our memoization system.

Key words: Memoization, function caching, object-oriented programming, software transactional memory

1 Introduction

Memoization, also known as function caching, was first introduced in 1968 [1] in the context of Artificial Intelligence as a way for machines to learn from past experiences, as if programs could "recall" previous computations and thus avoid repeated work. The key idea behind it is that we may speedup the execution of a function if we maintain a cache of previous computations and return results from that cache instead of computing the results again.

Memoization is often used in functional programming to increase the performance of a program, but is seldom used in other programming paradigms, such as object-oriented programming. The fundamental reason for this is that traditional memoization is limited to functions that produce no side-effects and do not access any shared mutable state. Thus, this limitation rules out most of the code that is used in traditional object-oriented programming.

In this work we describe the Automatic Transaction-Oriented Memoization (ATOM) system, which takes advantage of an implementation of a Software

* This work was partially supported by the Pastramy project (PTDC/EIA/72405/2006).

Transactional Memory (STM) for Java to extend the applicability of memoization to object-oriented programs.

To the best of our knowledge, ATOM is the first automatic memoization system that has any of the following benefits: (1) it allows memoizing methods that access shared state (rather than pure functions that depend only on their arguments), (2) it prevents errors in memoizing unintended methods, and (3) it allows memoizing methods that have side-effects. We argue that our extended memoization system is the first memoization system to suit the needs of object-oriented programming, and that it may come almost for free for systems that already use an STM.

In the next section we review what memoization is and what its major limitations are. Then, in Section 3, we introduce the concept of Software Transactional Memory, with special emphasis on the Java Versioned STM. In Section 4, we present the core ideas of our work, and then we describe its implementation in Section 5. In Section 6, we discuss how memoization can be particularly useful when combined with an STM and present some results obtained from the STMBench7 benchmark. In Section 7, we discuss related work, and, finally, in Section 8, we draw some conclusions and discuss future work.

2 Memoization

Memoization is a well-known technique for improving the performance of programs in exchange of extra memory space: To avoid future redundant work, each memoized function is augmented with a cache that stores the results of previous computations.

Unmemoized functions usually take some values as arguments and return another value that is computed from those arguments, regardless of whether that same computation was already done before. A memoized function, on the other hand, first checks whether it has already performed the asked computation in the past by searching its cache with the currently supplied arguments. If a mapping is found, it returns the stored result to the caller. If it is the first time that such computation is done, the requested result is computed and, just before its return to the caller, a new mapping is added to the cache representing this new arguments-to-result association.

A performance boost results when it is faster to search the cache for a match than to reexecute the function, and, of course, there is a cache hit (meaning that the result stored in the cache is returned). This gain in performance must be sufficient to compensate for possible cache misses and the cost of constructing and managing cache entries.

2.1 Limitations of memoization

Normally, memoization may be applied only to pure functions. So, it is not applicable to functions that are not deterministic, functions with behaviors that depend on the program's mutable state, or functions that have side-effects such as doing I/O or changing the program's state. Therefore, memoization is seldom applicable to methods of classes in an object-oriented program: Typically, these classes have internal state that affects the result of their methods.

```
public class Class1 {
  boolean slot;

  public int method1(int arg1, boolean arg2) {
    int res = /* Expensive computation */
    return res;
  }
}
```

Listing 1: Typical class in Java.

To better explain the limitations of applying memoization to object-oriented programs, we will use the simple example of a Java class shown in Listing 1.

If we want to memoize `method1`, we may insert a map in `Class1` to store the memo information and change the body of `method1` to search the map before computing the intended result. This is safe as long as the "expensive computation" expression is purely functional. But what happens if that expression accesses `slot`? For instance, consider the refinement of `method1` depicted in Listing 2.

```
public int method1(int arg1, boolean arg2) {
   int res = (slot
              ? /* Something expensive */
              : /* Something else expensive */);
   return res;
  }
```

Listing 2: Example of a method that returns a result that is influenced by an object's state—the value of `slot` of class `Class1`.

Assuming that the value of `slot` may change over time, it is clear that the simple solution of just using the arguments of `method1` to search in the memo cache is not enough: The value of `slot` is relevant to determine which value to return from `method1`, also.

A naive approach would be to extend the memo cache to include the value of `slot` in the keys of the cache map, too. In fact, in this simple example that may solve the problem. Yet, in the general case of more complex methods where the set of fields accessed is much larger, depends on which execution path is taken or on what other methods are called, it is unfeasible to determine manually the correct set of fields to use in the cache. This is one of the difficulties of applying memoization to object-oriented programs: Methods often depend on the state of other objects, many of which cannot be easily determined by code inspection alone. Still, in this example, `method1` just reads values from the objects' state.

Consider now the method depicted in Listing 3. This method behaves almost like `method1`, but in addition to `method1`'s behavior it toggles the value of `slot`. A simple memoization solution cannot be applied here because `method2` is not referentially transparent: Calling `method2` does not have the exact same effect

as replacing the method call with its return value. Again, methods such as this prototypical example are common in object-oriented programs.

```
public int method2() {
  int res = /* Expensive computation */
  slot = !slot;
  return res;
}
```

Listing 3: Example of a method that changes an object's state and that, thus, cannot be memoized.

3 Software Transactional Memory

Software Transactional Memories (STM) [2] introduced the concept of transaction in mainstream programming, mostly as a way to simplify concurrent programming. The key idea behind STMs is that programmers specify which operations must execute atomically, leaving to the transactional framework the responsibility of providing the intended atomicity semantics, while maintaining as much parallelism as possible.

From the perspective of an STM, operations executed within a transaction do not have a special meaning associated with them: They are just a series of reads from and writes to shared memory locations. STMs intercept these accesses to shared memory locations, so that they may detect when two concurrent transactions are interfering with one another, in which case the STM stalls, aborts, or restarts at least one of the transactions.

There are numerous STM implementations, varying considerably in how they ensure the atomicity of operations. Therefore, we will concentrate our discussion in a particular STM implementation, the Java Versioned STM (JVSTM) [3,4]. JVSTM is a multi-versioned STM implemented as a pure Java library[1] and it is the STM implementation that we use as a basis for the implementation of the ATOM system that we describe in this paper.

The JVSTM introduces the concept of *versioned boxes* as a replacement for shared memory locations. Conventional memory locations keep only a single value but a versioned box is a container that holds a tagged sequence of values representing all of the changes made to that location by some transaction.

To detect conflicts among concurrent transactions, JVSTM logs into a per-transaction read-set which boxes, and respective versions, were read inside that transaction. Likewise, it logs into a write-set which boxes were written and with what value; only at commit-time will these values be effectively written to the boxes, and only if the transaction is valid. These read-sets and write-sets are the key elements that will allow us to extend the applicability of memoization.

4 Extending the applicability of memoization

To extend the applicability of memoization to object-oriented programs, as we saw in Section 2.1, we need to address two problems. First, we need to be able to

[1] The source code of the JVSTM is available at `http://web.ist.utl.pt/~joao.cachopo/jvstm/`

capture all of the relevant state used to compute a result, other than the values received as arguments of a method; typically, this state includes values belonging to the program's shared state. Second, we need to identify which methods have side-effects, so that we may choose either to not memoize them or to collect sufficient information to correctly replicate their behavior. We shall show in this section how STMs can help to achieve both goals.

4.1 Finding all of the relevant state for memoization

To overcome the limitations of memoization regarding shared state accesses, it is first necessary to understand what is the relevant state for a particular method's execution. We define relevant state as all the values that are significant for obtaining a particular result so that if at least one of those values change the result may be different.

Recalling the example of `method1` shown in Listing 2, the relevant state for the result returned by `method1` includes the value `slot`, besides, presumably, the values of `arg1` and `arg2`. The problem, as we discussed then, is finding all of the relevant state in more complex cases.

To solve this problem, we propose to use the support already provided by an STM. If `method1` executes inside an STM transaction, then all of the memory read operations made within that transaction will be registered in the transaction's read-set. Thus, at the end of the transaction we will know which values were read to compute the method's result, thereby capturing all of the relevant state for the computation of this particular result.

4.2 Identifying side-effects

If a method that we intend to memoize has side-effects, we may adopt one of the two following approaches:

- Offer the conventional semantics associated with memoization and prohibit the memoization.
- Extend the concept of memoization to imperative methods, so that the execution of a memoized method reproduces the side-effects that it would produce if it was not memoized.

Obviously, the first approach is simpler to implement. Still, it is traditionally left to the responsibility of the programmer to decide when to memoize or not a given method, which is error-prone and suboptimal (because the programmer may decide not to memoize a given method because it may produce side-effects, even though most of the times it does not).

Leveraging again on the support given by an STM, we argue that it is possible to automate this approach with negligible costs: If we execute a method inside an STM transaction, once it finishes, we may look at the transaction's write-set to see whether the method wrote to any shared state; if it did, then we do not memoize this call; otherwise, we may memoize it as before. There is a subtle detail here which is worth mentioning. With this approach we are not identifying whether methods as a whole produce side-effects or not. Rather, we

are identifying, on a per-call basis, whether that call produces side-effects or not. Also, we are relying on the STM to be able to identify all of the possible side-effects, which may exclude side-effects such as doing I/O that are an open issue still in the area of STM research.

We turn our attention now to the second approach identified above of being able to memoize method calls with side-effects. Going back to the example shown in Listing 3, what does it mean to reproduce the behavior of `method2`, which has side-effects?

The externally observable behavior of `method2` is the return of its result and the change made to `slot`. So, to reproduce the behavior of `method2` it is necessary to replicate its return value and, also, correctly change the value of `slot`.

The answer comes again from the STM. Looking at the write-set, we may see which boxes were written and with what value. So, it is possible to memoize `method2` if we store the write-set in the cache and in the future, after a cache hit, we iterate over the associated write-set and reapply all the changes as an additional step of the memoization process.

5 The ATOM system

To implement our STM-based approach to memoization of object-oriented programs, we developed the Automatic Transaction-Oriented Memoization (ATOM) system. ATOM encapsulates memoized methods inside transactions to capture all of the relevant state for a particular result and to register possible write operations, so that it may apply them in future reexecutions of the same method.

5.1 The memo cache

For each memoized method, a private instance of the class `MemoCache` must be added to the respective class. The rational behind this decision is that we consider that, most often, the receiver of the message (the instance on which the method is being invoked on) influences the outcome of the method. Thus, by spreading the cache by all of the objects of a class, rather than having a single cache for all of them, naturally partitions the cache and simplifies its maintenance, because when an object is garbage-collected, so is the portion of the cache that belongs to it.

The `MemoCache` instance is responsible for providing all the memoization semantics. It assumes that there is an active transaction associated with the current method's execution and it is organized in two levels as show in Figure 1.

We use a table lookup to search the first level of the `MemoCache`, using the supplied arguments as the key. Once we obtain a second level entry, the search algorithm iterates over all its entries looking for a valid read-set. A read-set is valid if and only if all the boxes in it still hold the same value as when the read-set was stored in the cache.

To see whether a box still has the same value as before, we experimented with two caching policies. The first policy, implemented by `VersionMemoCache`, considers that a box still has the same value if and only if it is still in the

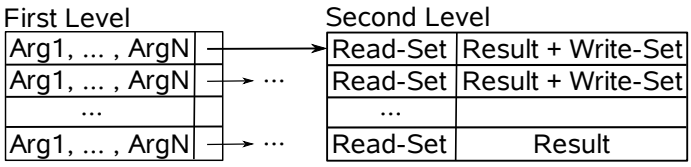


Fig. 1: Organization of the `MemoCache`. The first level is composed of all the information available at call time and maps to a second level which holds information only observable at runtime—that is, the captured read-set, the returned result, and, possibly, a write-set.

same version (meaning that no write occurred to this box). The second policy, implemented by `ValueMemoCache`, checks whether the current value of the box, regardless of its version, is equals to the value seen before; for this approach, we assume that the method `equals` is correctly defined for the values stored in the boxes.

5.2 The ATOM API

The `MemoCache` class, shown in Listing 4, constitutes the only visible interface for programmers.

```
public class MemoCache {
  public Object search(List<Object> args) { ... }
  public Object collectInformation(List<Object> args,
                                   Object res) { ... }
}
```

Listing 4: Skeleton of the class `MemoCache`. The two methods shown are the basic operations needed to memoize methods with the ATOM system.

The method `search` receives a list of arguments and is responsible for searching the cache for a hit. If the table lookup fails or none of the read-sets are valid, a `NotFoundException` is thrown. Otherwise, the associated result is returned and, if necessary, the stored write-set is applied.

Responsible for collecting all the relevant information about a method's execution, the method `collectInformation` receives a list with the arguments passed to the original method, the result the original method will return, and it is responsible for extracting the read-set (and write-set) from the associated transaction.

Because there is a very clear pattern to memoize methods, the ATOM system provides a method annotation to simplify the process: the `Memo` annotation.

Classes with methods that use this annotation are post-processed and rewritten, simplifying the task of memoizing methods: Programmers just need to express their intention, as shown in Listing 5, and the system automatically does the appropriate changes (shown in Listing 6).

As there are two possible caching policies, the `Memo` annotation may be parameterized with the appropriate `type`: `version` (which is the default) or `value`.

```
@Memo
int method1(int arg1, boolean arg2) {
  /* method1's Body */
}
```

Listing 5: Use of the annotation `Memo` to memoize the method `method1`. The necessary code is introduced by post-processing the bytecode, originating the code shown at Listing 6.

6 Memoizing transactions

Just like any other program, STMs could benefit from memoization, but we will see that applying memoization in a transactional system can be particularly useful.

In the JVSTM, as in most nonblocking STMs, changes made during a transaction are made permanent only at transaction's end (commit time) and only when the transactional system can assure that there are no consistency violations. So, the commit of a transaction is responsible for detecting conflicts and can yield one of two possible results:

- success—all of the values written by the transaction were applied to the shared system state.
- fail—none of the values written were applied and the transaction should be restarted.

So, STMs introduce overheads in the form of intercepted memory accesses, extra memory to store transactional information, time spent validating transactions, and in the reexecution of conflicting transactions.

We argue that memoization can be very useful in this context for at least two reasons. First, as it uses information that is already generated by the STMs, it amortizes that cost by speeding up memoized methods with no significant added cost. Second, because it can accelerate the reexecution of conflicting transactions by avoiding repeated computations.

6.1 Case study: the STMBench7 benchmark

The STMBench7 benchmark [5] is a highly customizable benchmark, developed for evaluating STM implementations. Its data structure is similar to CAD programs, consisting of atomic parts, assemblies, composite parts, connections, documents, manuals, and modules.

There is a single module connected to a deep trinomial tree of assemblies. Base assemblies, situated at the bottom-level, are comprised of several composite parts, which contain a document and a graph of atomic parts. This graph is connected through connection objects allowing bottom-up as well as top-down traversals.

The STMBench7 benchmark allows for additional synchronization strategies to be implemented. So, a new one was introduced that wraps each operation with a JVSTM transaction. To test the effectiveness of the memoization approach, we made a first run of the benchmark with all of the 14 operations of the benchmark

```
private MemoCache method1_I_Z$cache;

public int method1(int arg1, boolean arg2) {
  List<Object> args = new ArrayList<Object>(2);
  args.add(arg1);
  args.add(arg2);

  Transaction.begin();
  try {
    int res = method1_I_Z$cache.search(args);
    Transaction.commit();
    return res;
  } catch (NotFoundException e) {
    /* First Execution, so fall-through to execute method */
  }

  int res = method1_I_Z$memoized(arg1, arg2);
  method1_I_Z$cache.collectInformation(args, res);
  Transaction.commit();
  return res;
}

private int method1_I_Z$memoized(int arg1, boolean arg2) {
  /* original method1's body */
}
```

Listing 6: Memoized version of `method1`, after being post-processed.

annotated with `@Memo(type="version")`. Then, we selected the operations with the higher cache-hit rate and performance boost, which resulted in the operations `Query2`, `Query7`, `ShortTraversal1`, `Traversal1` and `Traversal8`. Then, we ran a series of tests with these selected operations memoized to obtain the results presented below.

Each test ran for 120 seconds in a Dual-Quadcore Intel Nehalem-based Xeon E5520, with 12Gb of RAM running Ubuntu Linux 9.04, and Java SE version 1.6.0_13.

We present results for the JVSTM and for the JVSTM with ATOM, using 1, 2, 4, 8, and 16 threads. All tests were for a read-dominated workload and with one of three possible mixes of operations: (1) all operations except long read-write traversals and structural modifications, (2) all operations except long traversals (both read-write and read-only), and (3) all operations except long traversals and structural modifications.

6.2 Experimental results

The results obtained for the three mixes of operations are shown in Figure 2 through 4.

These results show a clear increase in performance when using memoization in these test cases: The memoized version performs better than the plain

JVSTM in almost all scenarios, achieving the best results in the first mix of operations (shown in Figure 2), where the throughput of the system increases by a factor of 3. The first mix of operations includes long read-only traversals, which are the most computationally intensive operations in the benchmark—the maximum time to completion of long traversals is around 1 second, whereas for the remaining operations is below 10 milliseconds. So, it makes sense that memoization gives the best results for long-traversals.

These results are even more expressive if we take into account the fact that the run of JVSTM without memoization uses optimized read-only transactions that do not log reads in a read-set, whereas the memoized version needs to compute a read-set to be able to memoize the operations. Thus, this means that the benefits of memoization pay off the cost of computing very large read-sets in this case.

Still, even if we disable all of the expensive traversals, leaving only short operations in the mix, the results depicted in Figure 3 and 4 show that memoization behaves better then the JVSTM without memoization, demonstrating clear advantages of using memoization even for operations that are not computationally demanding. The use of memoization in the second mix almost doubles the throughput of the system for 4 threads.

The only case where the memoized version performs worse than the plain version is when we have 16 threads in the third mix, in which case both versions reduce their throughput. This result is typical of the STMBench7 when we have more threads than available processors, because the number of conflicts rise and so do the number of restarted transactions.

Overall, these results show that memoization is able to improve significantly the performance of read-dominated workloads. Our solution scales almost perfectly and given that the STMBench7 benchmark traverses the object graph but performs no operations on the leaves, it is reasonable to expect better results under a more realistic test.

7 Related work

Memoization has been the subject of investigation since it was first introduced in 1968. Many automatic memoization systems were introduced through the years for programming languages such as LISP [6] or C++ [7]. Compared to previous implementations of memoization, our solution is the first automatic memoization system that allows shared state dependencies, that validates choices made by the programmers, and that incorporates side-effects within the memoization process.

To identify functional methods in Java, Xu et al [8] proposed a dynamic solution based on escape and Java bytecode analysis. They applied memoization to weakly pure methods—those that only read shared state if it can be reached from supplied arguments and perform write operations as long as it is not to shared state. Even though it is an improvement over systems that memoize only pure functions, this approach is still too limited to be applied to object-oriented programs.

Ziarek and Jagannathan [9] first proposed the use of memoization to prevent the reexecution of operations that do not conflict, but their work assumed a

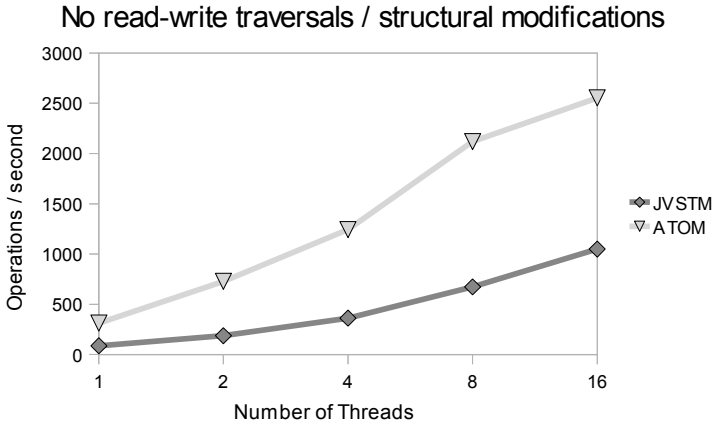


Fig. 2: Operations per second processed by the JVSTM with and without memoization, for a read-dominated workload with all long read-write traversals and structural modifications disabled.

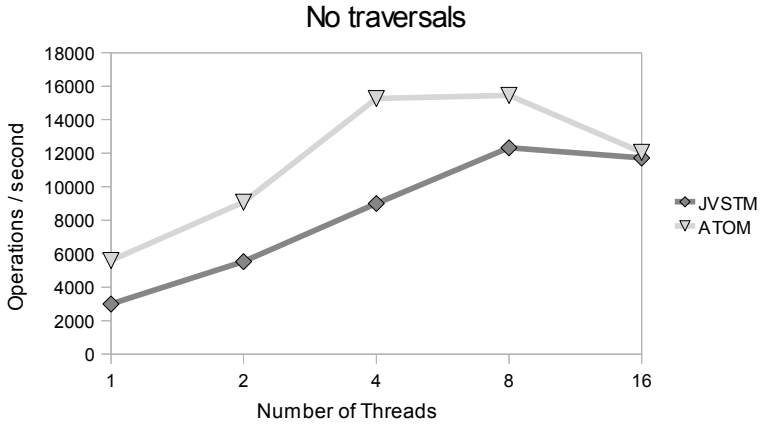


Fig. 3: Operations per second processed by the JVSTM with and without memoization, for a read-dominated workload with all long traversals disabled.

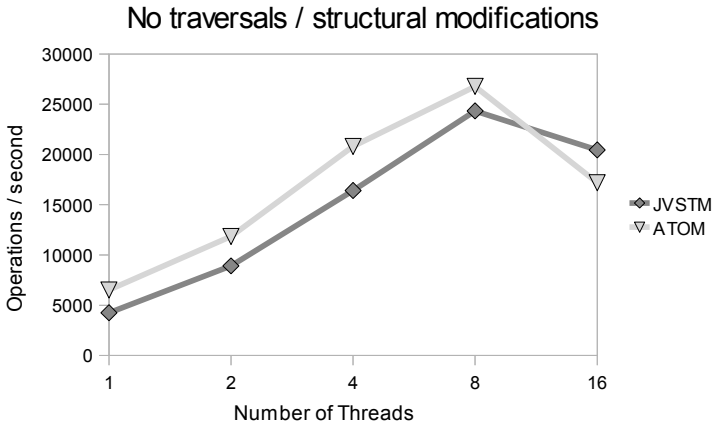


Fig. 4: Operations per second processed by the JVSTM with and without memoization, for a read-dominated workload with all long traversals and structural modifications disabled.

functional environment and only applies memoization to aborted transactions. By contrast, our work imposes no constraints on the characteristics of the system and allows the application of memoization to any kind of transaction.

8 Conclusions and future work

In this paper we proposed to use Software Transactional Memories to extend the applicability of memoization. This extended memoization system allows for shared state dependencies, programmers' intent validation, and even the incorporation of side-effects in the memoization process.

Our solution is to wrap methods with STM transactions to obtain all the information that is needed for the memoization system. This information comes from the STM, that already collects all the relevant information, which means that this extended memoization approach comes for free for a system that already uses an STM. Moreover, because it increases the performance of the system at almost no extra cost, it amortizes the upfront cost of using an STM, thereby promoting the adoption of STMs.

We implemented this approach in Java, as an extension to the Java Versioned STM, and discussed how memoization can help improve the performance of nonblocking STMs, specially in read-dominated workloads where the tests that we performed show an improvement of over three times the original throughput of the system.

A promising application of our memoization approach is in the restarting of conflicting transactions, if we memoize most of the methods ran within a transaction. This is an area that we intend to pursue in the future, as well as experimenting with other benchmarks.

References

1. Michie, D.: Memo functions and machine learning. Nature **218**(1) (1968) 19–22
2. Shavit, N., Touitou, D.: Software transactional memory. In: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing, ACM Press New York, NY, USA (1995) 204–213
3. Cachopo, J., Rito-Silva, A.: Versioned boxes as the basis for memory transactions. Science of Computer Programming **63**(2) (2006) 172–185
4. Cachopo, J.: Development of Rich Domain Models with Atomic Actions. PhD thesis, Technical University of Lisbon (September 2007)
5. Guerraoui, R., Kapalka, M., Vitek, J.: STMBench7: A Benchmark for Software Transactional Memory. ACM SIGOPS Operating Systems Review **41**(3) (2007) 315–324
6. Hall, M., Mayeld, J.: Improving the performance of AI software: Payoffs and pitfalls in using automatic memoization. In: Proceedings of the Sixth International Symposium on Artificial Intelligence. (1993) 178–184
7. McNamee, P., Hall, M.: Developing a tool for memoizing functions in C++. ACM SIGPLAN Notices **33**(8) (1998) 17–22
8. Xu, H., Pickett, C., Verbrugge, C.: Dynamic purity analysis for java programs. In: Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, ACM Press New York, NY, USA (2007) 75–82
9. Ziarek, L., Jagannathan, S.: Memoizing multi-threaded transactions. In: Workshop on Declarative Aspects of Multicore Programming. (2008)

Comparison of XAML and C# Forms using Cognitive Dimension Framework

Marjan Mernik[1], Tomaž Kosar[1], Matej Črepinšek[1], Pedro Rangel Henriques[2], Daniela da Cruz[2], Maria João Varanda Pereira[3] and Nuno Oliveira[2]

[1] University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova 17, 2000 Maribor, Slovenia
Email: {marjan.mernik, tomaz.kosar, matej.crepinsek}@uni-mb.si
[2] University of Minho - Department of Computer Science, Campus de Gualtar, 4715-057, Braga, Portugal
Email: {prh, danieladacruz, nunooliveira}@di.uminho.pt
[3] Polytechnic Institute of Bragança
Campus de Sta. Apolónia, Apartado 134 - 5301-857, Bragança, Portugal
Email: mjoao@ipb.pt

Abstract. Many domain-specific languages arise in the past years, trying to bring feasible alternatives for existing solutions with purpose to simplify programmers work. Although these little languages seem to be easier to use, there is an open issue whether they bring advantages comparing to most commonly used implementation approach, application libraries. In this work we present an experiment, carried out to compare such domain-specific language with comparable application library. The experiment was conducted with 36 programmers, which were answering questions on more than 100 long pages on both implementation approaches. For domain-specific language and application library the same problem domain has been used – construction of graphical user interfaces. In terms of domain-specific language, XAML has been used and C# Forms for application library. For comparison of XAML and C# Forms cognitive dimension framework has been used.

1 Introduction

The primary goal of developing new programming language is to make programming more efficient. The perfect programming language should provide the right level of abstraction, meaning that it describes solutions naturally and hides unnecessary details. Also, it should be expressive enough in the problem domain and should provide guarantees on properties critical for the problem domain. It should also have precise semantics to enable formal reasoning about a program. With general-purpose languages (GPLs) this is hard to achieve, since they tend to be general with consequence on poor support for domain-specific notation. On the other hand, domain-specific languages (DSLs) can be designed in many problem domains to have exactly above properties. DSL is a language tailored to specific application domain that offers appropriate notations and abstractions

[12]. DSLs are more expressive and are easier to use than GPLs for the domain in question, with gains in productivity and maintenance costs [6], [10], [18].

Although, DSLs have proven their usefulness, GPLs together with application libraries (APIs) are still the most commonly used programmer's choice when preparing new solutions for their problems. One of the reasons that DSLs are not accepted among the practitioners, is the lack of DSLs' promotion. Further, studies that would point out the benefits of DSL over GPL solution are rare. In this paper we will use cognitive dimension framework (CDF) [8] to compare DSL and GPLs programs and to expose properties that are enhanced in the context of DSLs. The goal of the project[4] is to measure how easy is to understand programs written in DSLs than in GPLs. In this manner, experiment is conducted using questionnaires to measure programmers understanding of DSL and GPL programs on same problem domain, a construction of graphical user interfaces (GUIs). More precisely, with these questionnaires we try to confirm that DSL programs are easier to understand than GPL programs. This hypothesis is here defended with experiment under controlled environment, using direct observations of experiment evaluation model involving CDF.

The organization of this paper is as follows. Related work on DSLs, preparation of experiment and CDF is discussed in Section 2. Experiment skeleton, the identification of its main goals, and experiment details are the topics introduced in Section 3. Experiment results, with cognitive dimension framework, are given in Section 4. Concluding remarks with future work are summarized in Section 5.

2 Related work

The DSL is usually developed in the following phases: decision, analysis, design, implementation, and deployment as identified in work [12]. For the implementation part, following techniques have been identified for DSLs: preprocessing, embedding, compiler/interpreter, compiler generator, extensible compiler/interpreter, and commercial off-the-shelf. In order to implement a DSL, a programmer has to choose among these implementation approaches and, of course, the most suitable one should be chosen according to project influences. As defined above, one of the implementation techniques is also commercial off-the-shelf approach. Here, existing tools and/or notations can be used to a specific domain, e.g., XML-based DSLs. This approach provides a feasible alternative when solving particular domain problems and for instance, XML here brings promising solutions in processing and querying documents for data exchange, etc. In general, XML tends to be cumbersome for humans to read and write and has some other disadvantages comparing to other DSL implementation approaches [10], however it seems that the approach is very well accepted by the leading technologies in the software industry and it is not expected that this fact will change in the near future.

[4] This work is sponsored by bilateral project "Program Comprehension for Domain-Specific Languages" (code BI-PT/08-09-008) between Slovenia and Portugal.

This work can be classified to empirical software engineering category. Empirical research in software engineering is an important discipline, showing practical results on how practitioners (developers, end-users) come to accept and use technologies, techniques, etc. In order to avoid questionable results and to enable repetition of research giving the same results, experiment has to be prepared with caution. One of the most well known framework for software experiments is described in [2]. This framework concentrates on building knowledge about the context of an experiment and is based on organizing sets of related studies (family of studies). Such studies contribute to a common hypotheses which does not vary for individual experiments. Therefore, we followed guidelines from framework [2] in order to prepare this experiment and we defined: context of the study, experiment hypothesis, comparison validity, and measurement framework.

Teaching environments give us an opportunity to conduct experiments also in computer science programs. However, a lot of concerns are connected with the accuracy of results in such environments and several threats to validity of experiment has to be identified and to interpret the results correctly. Interested reader, can find more about this topic in the work [4], where a checklist for integrating empirical studies in teaching activities can be found.

As stated above, important step in experiment preparation is to set down the measurement framework – how results of experiment are evaluated and interpreted. In cognitive theory, guidelines how to measure human's ability to program, are defined. CDF [8] provides cognitively-relevant aspects which can be used to determine how easy it is to understand a program. In our study, CDF is used to compare user understanding of DSL and GPL programs. While before CDF has been used to assess the usability of visual programming languages [3], [17] and spreadsheets [13].

Another application of cognitive dimensions can be found recently in [1], where method for designing Framework-Specific Modeling Languages (FSMLs) is presented. From FMLS specifications user can build applications based on object-oriented frameworks. In FSML software artifacts (models, languages, etc.) are evaluated according to its goals with different quality methods. Particularly, quality of notation is measured with cognitive dimensions – a heuristic measure that evaluates the notation and its environment.

Before this experiment, authors of the paper were involved in another similar experiment [9]. That work is important for interested reader, since information on experiment skeleton is described in details. Difference among both experiments is in hypothesis and exclusion/inclusion of CDF. Also, the problem domain in experiment [9] is different (graph description with DOT language [7]) than in this paper (construction of GUI with XAML).

This paper is also closely related to the field of *Program Comprehension*, which is a hard cognitive task done by software analyst. In the process of program comprehension, the use of tools to interconnect different views (operational, behavioral, etc.) to understand results of application, are indispensable. Traditional techniques on program comprehension from GPLs (visualizers, animators,

etc.) have been studied and applied to DSLs in our previous work [15], where CDF was also briefly described and applied to DSLs.

3 Presentation of experiment

In this section the preparation, execution, and experiment evaluation model is given.

3.1 Subject of comparison

In the work [10] the empirical results comparing ten diverse implementation approaches for DSLs, conducted on the same representative language, are provided. Among implementation approaches, comparison included also XML-based approach. From this study can be concluded, that XML-based approach, has some disadvantages [10]. Although, XML usage and its tool support are spreading. This is one of the reasons that XAML [16], as a representative DSL, has been chosen for this study. XAML, the Extensible Application Markup Language, is a language for representation of graphical user interfaces in Windows Presentation Foundation and Silverlight applications of .NET Framework 3.5. C# Forms [5] has been used for comparison since it covers the same domain of graphical user interfaces.

3.2 Preparation of experiment environment

The results from an experiment are reliable if the repetition of experiment can be proven [14]. Repetition is strongly connected to agreements set down before starting the experiment [2]. Consistency of results in our experiment were obtained by creating rules and constraints for programmers: using well-structured questionnaires, domain tutorials and extra explanations in their native language, before starting answering on questions. Tutorial to programmers included presentation of problem domain (GUI), domain specific notation (XAML) and application library (C# Forms) together with examples of programs. Consistency of results were obtained also with rules for questionnaire implementors, which had to define the same group of questions for both experiment on GPL and DSL. Implementors were also advised to prepare questions for two applications (easier and harder application domain). More on preparation of this experiment can be found in [9].

3.3 Threats to validity of experiment

In each experiment, there are several threats to validity of results. Those threats needs to be identified and handled before starting the experiment. To restrict the impact of the experiment environment on the results, following issues have been identified for our study.

Chosen domain Results of the experiment are strongly connected to programmers' experiences and knowledge of chosen problem domain. In Table 1, programmers familiarity with construction of GUI is presented, together with experience on XAML and C# Forms library application. From Table 1, we can conclude that programmers are experienced in domain of construction GUIs. However, their experience in implementation technique differs – programmers were unfamiliar with XAML on one hand (median value 1), and had good knowledge in constructing GUIs with C# Forms (median value 4). Uneven knowledge on both notations could have influence on comparison results.

Table 1. Programmers knowledge in construction of graphical user interfaces (N = 36)

	Average[1]	**Median**	**St.dev.**
Familiarity GUI domain	**3.39**	**4**	**1.18**
Knowledge of XAML	**1.36**	**1**	**0.68**
Knowledge of C# Forms application library	**3.5**	**4**	**1.11**

N = number of received questionnaires

Programmers experience In Table 2, we present results from self evaluation test, where students (second year of undergraduate computer science) grade their general knowledge about programming, programming in C# language and prior experience with DSLs. Comparing knowledge on C# (median value 4) and prior experience with DSLs (median value 2) could also have influence on experiment results.

Comparability of questionnaires Same type of questions in DSL and GPL questionnaires should contain similar number of graphical components (labels, text fields, buttons, etc), to obtain the same level of complexity.

Experiment questionnaires As stated before, two questionnaires have been prepared for program understanding of DSL and GPL programs. Then, the structure of questionnaires has been defined to cover following three topics of program understanding: learn, perceive, and evolve. In the first group, questions on learning notation and meaning of programs have been given to the programmers. In the second group, questions on program perceiving have been defined, such as identification of: correct meaning from given program, language constructs, new construct meaning, and meaning of program with given comments. In the third

[1] A five-graded scale, going from very bad (1) to very good (5) was used for self-evaluation questionnaires (in Tables 1 and 2). Note, that column "Average" shows the average value given by 36 programmers, "Median" stands for middle value in set of programmers grades and "St. Dev." represents standard deviation on given grades.

Table 2. Programmers experiences in programming (N = 36)

	Average	Median	St.dev.
Skills in programming	**3.41**	**3.5**	**0.65**
Skills of programming in C#	**3.53**	**4**	**0.74**
Prior experience with DSLs	**2.28**	**2**	**0.70**

group, programmers had been challenged to expand/remove/replace program functionality.

For these three groups, 11 questions have been defined:

- Learn
  - Q1 Select syntactically correct statements.
  - Q2 Select program statements with no sense (unreasonable).
  - Q3 Select valid program with given result.
- Perceive
  - Q4 Select correct result for the given program.
  - Q5 Identify language constructs.
  - Q6 Select program with same result.
  - Q7 Select correct meaning for the new language construct.
  - Q8 Identify language constructs in the program with comments.
- Evolve
  - Q9 Expand the program with new functionality.
  - Q10 Remove functionality from the program.
  - Q11 Change functionality in the program.

Learning and perceiving questions has been defined as multiple choice question, and questions on evolve has been defined as essay question (programmers are challenged to modify existing code). Both, XAML and C# Forms questionnaires have been constructed using the above questions.

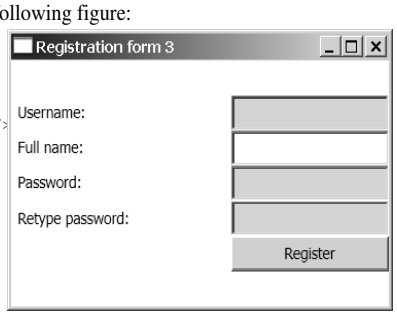
To illustrate the style of the questions used in the questionnaires, an example is presented in Figure 1. Because of question size only the correct choice is given. Complete questionnaires can be found at project group webpage[5].

4 Results

All together, programmers answered 22 questions on both questionnaires. Success rate for questions vary from 27.14% for Q6 to 79.73% for Q9 (Table 3). Differences in success rate in the same language (DSL/GPL) can be explained with different difficulty level (some questions were harder than others). On the other hand the biggest difference between GPL and DSL is 51.16% in case of Q9. The smallest difference we can find in Q2, where difference is just 3.44%. In this

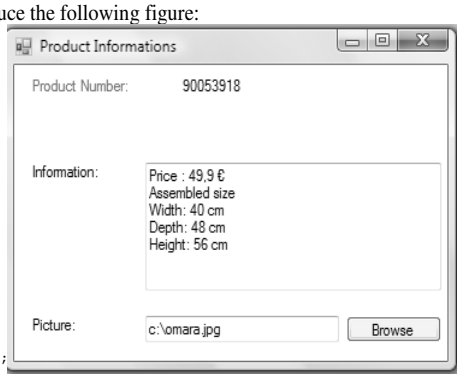
[5] http://epl.di.uminho.pt/~gepl/DSL/

Question 5
QL031 XAML-DSPL-RegistrationForm: Select program for the following figure:




```
<Window x:Class="WpfRegistration.Registration3"
  Title="Registration form 3" Height="200" Width="300">
  <Grid ShowGridLines="False">
    <Grid.ColumnDefinitions>
     <ColumnDefinition Width="12*"/> <ColumnDefinition Width="2*"/>
     <ColumnDefinition Width="10*"/> </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
   <RowDefinition Height="10*"/> <RowDefinition Height="10*"/>
   <RowDefinition Height="10*"/> <RowDefinition Height="10*"/>
   <RowDefinition Height="10*"/> <RowDefinition Height="10*"/>
   <RowDefinition Height="10*"/> </Grid.RowDefinitions>
   <Label   Grid.Column="0" Grid.Row="1"> Username:</Label>
   <TextBox Grid.Column="2" Grid.Row="1" Background="LightGray"/>
   <Label   Grid.Column="0" Grid.Row="2"> Full name:</Label>
   <TextBox Grid.Column="2" Grid.Row="2"/>
   <Label   Grid.Column="0" Grid.Row="3"> Password:</Label>
   <TextBox Grid.Column="2" Grid.Row="3" Background="LightGray"/>
   <Label   Grid.Column="0" Grid.Row="4"> Retype password:</Label>
   <TextBox Grid.Column="2" Grid.Row="4" Background="LightGray"/>
   <Button  Grid.Column="2" Grid.Row="5"> Register</Button> </Grid> </Window>
```

Question 5
QL031 WFORM-GPL Select the correct program to produce the following figure:




```
private Label labelNumber, labelFilledNumber;
private Label labelInf, labelPic;
private TextBox textBoxInfo, textBox1;
private Button btnBrowse;

private void InitializeComponent() {
  labelNumber = new Label();
  labelFilledNumber = new Label();
  textBoxInfo = new TextBox();
  labelInf = new Label(); labelPic = new Label();
  btnBrowse = new Button(); textBox1 = new TextBox();
  labelNumber.ForeColor = System.Drawing.Color.Red;
  labelNumber.Location = new System.Drawing.Point(12, 9);
  labelNumber.Name = "labelNumber";
  labelNumber.Size = new System.Drawing.Size(119, 17);
  labelNumber.TabIndex = 5;
  labelNumber.Text = "Product Number: ";
  labelFilledNumber.Name = "labelFilledNumber";
  labelFilledNumber.Size = new System.Drawing.Size(72, 17);
  labelFilledNumber.Text = "90053918";
  labelFilledNumber.Location = new System.Drawing.Point(137, 9);
  textBoxInfo.Location = new System.Drawing.Point(109, 74);
  textBoxInfo.Multiline = true; textBoxInfo.Name = "textBoxInfo";
  textBoxInfo.Size = new System.Drawing.Size(246, 97);
  textBoxInfo.Text = "Price : 49,9 €\r\nAssembled size\r\nWidth: 40 cm\r\nDepth: 48 cm\r\nHeight: 56 cm";
  labelInf.Location = new System.Drawing.Point(12, 74);
  labelInf.Name = "labelInf";
  labelInf.Size = new System.Drawing.Size(82, 17);  labelInf.Text = "Information:";
  labelPic.Location = new System.Drawing.Point(12, 190);
  labelPic.Name = "labelPic";
  labelPic.Size = new System.Drawing.Size(56, 17);  labelPic.Text = "Picture:";
  btnBrowse.BackColor = System.Drawing.Color.Yellow;
  btnBrowse.Location = new System.Drawing.Point(276, 190);
  btnBrowse.Name = "btnBrowse";
  btnBrowse.Size = new System.Drawing.Size(79, 20);  btnBrowse.Text = "Browse";
  btnBrowse.UseVisualStyleBackColor = false;
  textBox1.Location = new System.Drawing.Point(109, 190);
  textBox1.Name = "textBox1";
  textBox1.Size = new System.Drawing.Size(161, 20);
  Controls.Add(textBox1); Controls.Add(btnBrowse); Controls.Add(labelPic); Controls.Add(labelInf);
  Controls.Add(textBoxInfo); Controls.Add(labelFilledNumber); Controls.Add(labelNumber);
 }
```

Fig. 1. Question 5 in DSL and GPL questionnaires with correct choice

Table 3. Average programmer success rate (N = 36)

Question	DSL	GPL	Difference
	XAML	C# Forms	
Q1	72.97%	48.57%	24.4%
Q2	35.14%	38.57%	-3.44%
Q3	64.86%	35.71%	29.15%
Q4	77.03%	70.00%	7.03%
Q5	64.86%	48.57%	16.29%
Q6	39.19%	27.14%	12.05%
Q7	75.68%	62.86%	12.82%
Q8	62.16%	45.71%	16.45%
Q9	79.73%	28.57%	51.16%
Q10	68.92%	41.43%	27.49%
Q11	66.22%	30.00%	36.22%

Table 4. Average programmer success on learn, perceive and evolve (N = 36)

	Question	DSL	GPL	Difference
		XAML	C# Forms	
Learn	Q1, Q2, and Q3	57.66%	40.95%	16.71%
Perceive	Q4, Q5, Q6, Q7, and Q8	63.78%	50.86%	12.93%
Evolve	Q9, Q10, and Q11	71.62%	33.33%	38.29%

case success rate was even slightly better for GPL than DSL. To our opinion this is due to difficultness of Q2 (success rate was less than 39%), where syntactically correct programs with no sense have to be identified. Since programmers have more experience in C# Forms than XAML (Table 1) they were more successful for GPL than DSL in finding programs with no sense.

However, drawing conclusions based on average value of single question can be extremely risky. Therefore, by grouping questions on learn, perceive and evolve we can obtain more reliable results. In Table 4 average success rate on questions by individual group are presented. Table 4 confirms our presumption that program understanding in terms of learn, perceive and evolve is much better for DSL programs than on GPL programs. Later observation is specially obvious from results on evolve questions – success rate on this question was 38.29% better for DSL than on GPL questions. Similar results were also obtained on other problem domain described in [9].

One of possible explanation, why programs written in DSLs are easier to understand than programs written in GPLs, can be offered by CDF [8]. The CDF has been used before to assess the usability of visual programming languages [3], while no such study exists for DSLs. These cognitive dimensions are:

- Closeness of mapping – languages should be task-specific;
- Viscosity – revisions should be painless;
- Hidden dependencies – the consequences of changes should be clear;
- Hard mental operations – no enigmatic is allowed;
- Imposed guess-ahead – no premature commitment;
- Secondary notation – allow to encompass additional information;
- Visibility – search trails should be short;
- Consistency – user expectations should not be broken;
- Diffuseness – language should not be too verbose;
- Error-proneness – notation should catch mistakes avoiding errors;
- Progressive evaluation – get immediate feedback;
- Role expressiveness – see the relations among components clearly;
- Abstraction gradient – languages should allow different abstraction levels.

The next step was to connect cognitive dimensions with our questions. We identified which dimensions are relevant for particular question (Table 5). As it can be seen Di (dimension i of CDF) can be related with several questions used in our questionnaires.

Table 5. Questions connection to cognitive dimensions

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Closeness of mapping	1	1	1	1	1	1	1	1	1	1	1
Viscosity	0	0	0	0	0	0	0	0	1	1	1
Hidden dependencies	0	0	1	1	1	1	0	1	0	1	0
Hard mental operations	0	1	1	1	1	1	1	1	0	0	0
Imposed guess-ahead	0	0	0	0	0	0	0	0	1	0	1
Secondary notation	0	0	0	0	0	0	0	1	0	0	0
Visibility	0	0	1	1	1	1	1	1	0	0	0
Consistency	0	0	0	0	0	1	1	0	0	0	0
Diffuseness	1	1	1	1	1	1	1	1	1	1	1
Error-proneness	1	1	1	1	1	1	1	1	1	1	1
Progressive evaluation	0	0	0	0	0	0	0	0	0	0	0
Role expressiveness	0	1	1	1	1	1	1	1	1	1	1
Abstraction gradient	0	0	1	1	1	1	1	1	0	0	0

To evaluate single cognitive dimension (Di), the following formula has been used:

$$D_i = \sum_{j=1}^{11} Q_{ij} * \frac{S_j}{C_j}$$

where Q_{ij} stands for value from Table 5 meaning whether dimension Di is connected with question Q_j. Variable S_j represents average programmers success rate on question Q_j (Table 3). For example, if 4 programmers out of 5 answered correctly on question Q_1, the value of S_1 would be 0.8. Finally, C_j represents the

number of cognitive dimensions relevant for Q_j (for example, $C_1 = 3$). This formula is used for XAML as well as for C# Forms. Intuitively, it means that cognitive dimensions contribute to the success of particular question. Here, we assume that contribution of involved cognitive dimensions were equally distributed (one cognitive dimension is not more important than other if it is involved). Moreover, we assume that higher values always mean positive influence of particular cognitive dimension. For example, higher values for 'closeness of mapping' mean that the semantic gap between the problem and solution space is small, or higher values for 'hidden dependencies' mean that short and long-range interactions among program components are immediately visible.

Table 6 roughly shows how much particular cognitive dimension contribute to the questionnaires' success for XAML, as well as for C# Forms. From Table 6 it can be seen that in our experiment the most influential for DSL/GPL program understanding were: closeness of mappings, diffuseness, error-proneness, role expressiveness, and hard mental operations. More than particular values the important is difference among cognitive dimensions for XAML and C# Forms. The biggest difference among cognitive dimensions were at closeness of mappings, diffuseness, error-proneness, role expressiveness, and viscosity.

Table 6. Influence of cognitive dimension to XAML and C# Forms

	DSL	**GPL**	**Difference**
	XAML	**C# Forms**	
Closeness of mapping	**1.127**	**0.749**	**0.377**
Viscosity	**0.442**	**0.237**	**0.206**
Hidden dependencies	**0.486**	**0.343**	**0.143**
Hard mental operations	**0.525**	**0.421**	**0.105**
Imposed guess-ahead	**0.243**	**0.098**	**0.146**
Secondary notation	**0.069**	**0.051**	**0.018**
Visibility	**0.455**	**0.344**	**0.111**
Consistency	**0.128**	**0.100**	**0.028**
Diffuseness	**1.127**	**0.749**	**0.377**
Error-proneness	**1.127**	**0.749**	**0.377**
Progressive evaluation	**N/A**	**N/A**	**N/A**
Role expressiveness	**0.884**	**0.587**	**0.296**
Abstraction gradient	**0.455**	**0.344**	**0.111**

Closeness of mapping refers to how wide the semantic gap is between the problem and solution spaces. Diffuseness refers to the number of symbols needed to express the meaning. By definition, DSLs use existing domain notation which should be at an appropriate level of verbosity, so it is expected that they exhibit low diffuseness. On the other hand, it was shown in the study [11] that plenty of low-level primitives, which are often purely syntactical, is one of the biggest

cognitive barriers for end-user programmers. Error proneness refers to the capability of a language to induce careless mistakes. GPLs, due to their extension and intrinsic complexity, are usually error-prone. While, DSLs due to the narrow domain they are designed for, are usually less error prone. Role expressiveness refers to the ability to see how each component of a program relates to the whole. The high role expressiveness can be more easily achieved in DSLs due to domain specifics and shorter programs. It is shown in our experiment that differences of closeness of mapping, diffuseness, error proneness, and role expressiveness among XAML and C# Forms are the biggest and the source of main contribution for easier understanding of XAML programs than programs written in C# Forms.

Viscosity refers to how much effort is needed to perform small changes. Since DSLs are usually at high abstraction level and have natural notation small changes should be easier to perform. It is shown in our experiment that the difference in viscosity among XAML and C# Forms was among the biggest. Viscosity was involved only in questions Q9-Q11, which were much better solved using XAML and C# Forms. We can conclude that viscosity had an important influence to this success.

5 Conclusion and future work

The purpose of this paper is to promote formal studies on advantages of DSLs over GPLs. In this paper we tried to explain the difference among DSL/GPL program understanding using cognitive dimension framework. Questionnaires on understanding programs have been prepared and given to the programmers. Each programmer answered questionnaires written in more than 100 pages and on average spent more then 3 hours solving 44 questions.

Results show that programmers success rate was around 15% better for DSL in all three groups of questions: learn, perceive and evolve, despite that programmers were significantly less experienced in XAML than C# Forms. Further, experiment measurement framework included cognitive dimensions to identify the aspects among these dimensions that are enhanced in the context of DSL. From the study can be learned that DSLs gain comparing to GPLs in all cognitive dimensions. The cognitive dimensions with the biggest influence in the experiment are: closeness of mappings, diffuseness, error-proneness, role expressiveness, and viscosity.

We consider that the results of this experiment are reliable despite that experiment has been done only on single domain. One of the future tasks of this project, is to commit similar experiments in different domains.

References

1. Antkiewicz, M., Czarnecki, K., Stephan, M. *Engineering of Framework-Specific Modeling Languages*, To appear in IEEE Transactions on Software Engineering, 2009, http://doi.ieeecomputersociety.org/10.1109/TSE.2009.30.

2. Basili, V., Shull, F., Lanubile, F. *Building Knowledge through Families of Experiments*, IEEE Transactions on Software Engineering 25(4) 456–473, 1999.
3. Blackwell, A.F. *Ten years of cognitive dimensions in visual languages and computing: Guest editor's introduction to special issue*, Journal of Visual Languages and Computing 17(4), 285–287, 2006.
4. Carver, J., Jaccheri, L., Morasca S., Shull F., *A Checklist for Integrating Student Empirical Studies with Research and Teaching Goals*, To appear in Empirical Software Engineering, doi: 10.1007/s10664-009-9109-9.
5. C# Windows Forms, *Available at:* http://en.wikipedia.org/wiki/Windows_Forms
6. Deursen, A. van, Klint, P. *Little languages: Little maintenance?*, Journal of Software Maintenance (10), 75–92, 1998.
7. Dot – Graph Description Language, *Available at:* http://en.wikipedia.org/wiki/DOT_language
8. Green, T., Petre, M. *Usability analysis of visual programming environments: a "cognitive dimensions" framework*, Journal of Visual Languages and Computing 7(2) 131–174, 1996.
9. Kosar, T., Mernik, M., Črepinsek, M., Henriques, P. R., Cruz, D. da, Varanda Pereira, M. J., Oliveira, N. *Influence of domain-specific notation to program understanding*, Submitted to 2nd Workshop on Advances in Programming Languages (WAPL'09), 2009.
10. Kosar, T., Martínez López, P. E., Barrientos, P. A., Mernik, M. *A Preliminary Study on Various Implementation Approaches of Domain-Specific Language*, Information and Software Technology 50(5) 390–405, 2008.
11. Lewis, C., Olson, G. *Can principles of cognition lower the barriers to programming?*, 2nd workshop on Empirical Studies of Programmers, 1987.
12. Mernik, M., Heering, J., Sloane, A. *When and How to Develop Domain-Specific Languages*, ACM Computing Surveys 37(4) 316–344, 2005.
13. Peyton Jones, S., Blackwell, A., Burnett, M. *A User-Centred Approach to Functions in Excel*, Proceedings of the eighth ACM SIGPLAN international conference on Functional programming, 165–176, 2003.
14. Shull, F., Carver, J., Vegas, S., Juristo, N., *The Role of Replications in Empirical Software Engineering*, Empirical Software Engineering 13(2) 211–218, 2008.
15. Varanda Pereira, M. J., Mernik, M., Cruz, D. da, Henriques, P. R., *Program Comprehension for Domain-Specific Languages*, Journal on Computer Science and Information Systems, 5(2) 1–17, 2008.
16. XAML – Extensible Application Markup Language, *Available at:* http://en.wikipedia.org/wiki/Extensible_Application_Markup_Language
17. Yang, S., Burnett, M., DeKoven, E., Zloof, M. *Representation design benchmarks: a design-time aid for VPL navigable static representations*. Journal of Visual Languages and Computing 8(5/6) 563–599, 1997.
18. Živanov, Ž., Rakić, P., Hajduković, M. *Using Code Generation Approach in Developing Kiosk Applications*, Journal on Computer Science and Information Systems, 5(1) 41–59, 2008.

Iterators, Recursors and Interaction Nets

Ian Mackie[1], Jorge Sousa Pinto[2], and Miguel Vilaça[2]
mackie@lix.polytechnique.fr,{jsp,jmvilaca}@di.uminho.pt

[1] LIX, CNRS UMR 7161, École Polytechnique, 91128 Palaiseau Cedex, France
[2] Departamento de Informática / CCTC, Universidade do Minho, Braga, Portugal

Abstract. We propose a method for encoding iterators (and recursion operators in general) using interaction nets (INs). There are two main applications for this: the method can be used to obtain a visual notation for functional programs; and it can be used to extend the existing translations of the λ-calculus into INs to languages with recursive types.

1 Introduction

The use of visual notations for functional programs has long been an active research topic, whose main goal is to have a notation that can be used (i) to define functional programs visually, and (ii) to animate their execution.

In this paper we propose a graphical system for functional programming, based on token-passing INs [9]. The system offers an adequate solution for classic problems of visual notations, including the treatment of higher-order functions, pattern-matching, and recursion (based on the use of recursion operators). The system implements a call-by-name semantics, with a straightforward correspondence between functional programs and graphical objects.

Most approaches to visual programming simply propose a notation for programs. Program evaluation is animated by representing visually the intermediate programs that result from executing reduction steps on the initial program, using the operational semantics of the underlying functional language. Our approach is different in that we use a graph-rewriting formalism with its own semantics.

The advantages of using INs for visual programming are:

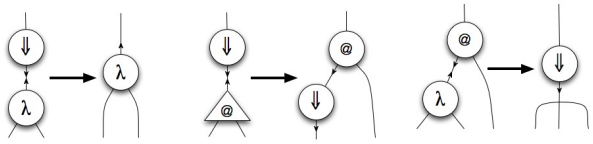
- Both programs and data are represented in the same graphical formalism.
- Programs can be animated without leaving the interaction formalism.
- Pattern-matching for external constructors is in-built.
- Recursive definitions are expressed very naturally as interaction rules involving agents that are reintroduced on the right-hand side.

But the interaction net formalism does not offer a satisfactory semantic interpretation for the behaviour of functional symbols. Moreover, many interaction net systems can be defined that do not have a functional reading. What is missing is a clear correspondence between functional definitions and interaction systems.In this paper we establish a correspondence between agents with "obviously functional" interaction rules and functions defined with recursion operators.

2 The Token-passing Encoding of the λ-calculus

The *token-passing* encodings [9] use an interaction system where two different symbols exist for application: one is the syntactic symbol @ introduced by the translation; the corresponding agents have their principal ports facing the root of the term and will be depicted by triangles. A second symbol $\widehat{@}$ exists that will be used for computation; to simplify the figures, the corresponding agents will be depicted by circles equally labelled with @. Their principal ports face the net that represents the applied function, to make possible interaction with λ agents.

The translation $\mathcal{T}_{\mathrm{tp}}(\cdot)$ encodes terms in the system $(\Sigma_{\mathrm{tp}}, R_{\mathrm{tp}})$ where $\Sigma_{\mathrm{tp}} = \{\Downarrow, @, \widehat{@}, \lambda, c, \varepsilon, \delta\}$. It generates nets containing no active pairs.The special symbol $\Downarrow$ is used as an evaluation *token*: an agent $\Downarrow$ traverses the net, transforming occurrences of @ into $\widehat{@}$. The evaluation rules involving $\Downarrow$ can be tailored to a specific evaluation strategy. For call-by-name, R_{tp} consists of the following rules which comprises evaluation rules involving $\Downarrow$ and a computation rule involving @ and λ. Management (copying and erasing) rules are omitted here.



To start the reduction a $\Downarrow$ symbol must be connected to the root port of the term. Let $\Downarrow N$ denote such net; then the following correctness result holds: $t \Downarrow z$ iff $\Downarrow \mathcal{T}_{\mathrm{tp}}(t) \longrightarrow^* \mathcal{T}_{\mathrm{tp}}(z)$, where the evaluation relation $\cdot \Downarrow \cdot$ is defined by the standard evaluation rules for call-by-name.

The language used in this paper is the simply-typed λ-calculus extended with natural numbers, booleans, lists, and iterators for these types. BNL is defined by the following syntax for terms (x, y range over a set of variables):

$$t, u, v ::= x \mid \lambda x.t \mid t\,u \mid \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{iterbool}(t, u, v) \mid 0 \mid \mathsf{suc}(t) \mid \mathsf{iternat}(\lambda x.t, u, v) \\ \mid \mathsf{nil} \mid \mathsf{cons}(t, u) \mid \mathsf{iterlist}(\lambda xy.t, u, v)$$

3 A Token-passing Encoding of BNL

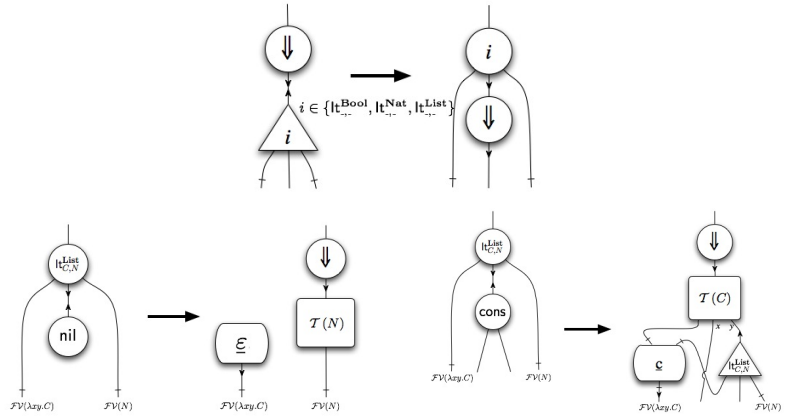
We extend to BNL the token-passing call-by-name translation of the λ-calculus into the interaction system $(\Sigma_{\mathrm{tp}}, R_{\mathrm{tp}})$. The novelty of this encoding is not the token-passing aspect, but rather the approach to recursion.

We first consider data structures. In a token-passing implementation, there will be an interaction rule between the token agent and each constructor symbol that will stop evaluation. For BNL we define the system $(\Sigma_{\mathrm{BNL}}, R_{\mathrm{BNL}})$ where Σ_{BNL} consists of the symbols tt, ff, 0 and nil with arity 0; suc with arity 1; and cons with arity 2. Each recursive program will be encoded in an interaction system specifically generated for it. This is a major novelty of our approach. The interaction system for the λ-calculus will not be extended by introducing a fixed set of symbols; instead a new symbol will be introduced for *each occurrence of*

a recursion operator, with an interaction rule for each different constructor, so a dedicated interaction system (Σ_t^0, R_t^0) is generated for each term t. This system is constructed by a recursive function $(\Sigma_t^0, R_t^0) = \mathcal{S}(t)$, defined as:

$\mathcal{S}(x) \doteq \mathcal{S}(\mathsf{tt}) \doteq \mathcal{S}(\mathsf{ff}) \doteq \mathcal{S}(0) \doteq \mathcal{S}(\mathsf{nil}) \doteq (\emptyset, \emptyset)$
$\mathcal{S}(\lambda x.t) \doteq \mathcal{S}(\mathsf{suc}(t)) \doteq \mathcal{S}(t)$
$\mathcal{S}(t\,u) \doteq \mathcal{S}(\mathsf{cons}(t,u)) \doteq \mathcal{S}(t) \cup \mathcal{S}(u)$
$\mathcal{S}(\mathsf{iterbool}(V,F,b)) \doteq (\{\mathsf{lt}^{\mathbf{Bool}}_{V,F}, \widehat{\mathsf{lt}^{\mathbf{Bool}}_{V,F}}\} \cup \Sigma, R_{\mathsf{lt}^{\mathbf{Bool}}_{V,F}} \cup R)$,
where $(\Sigma, R) = \mathcal{S}(b) \cup \mathcal{S}(V) \cup \mathcal{S}(F)$.
$\mathcal{S}(\mathsf{iternat}(\lambda x.S, Z, n)) \doteq (\{\mathsf{lt}^{\mathbf{Nat}}_{S,Z}, \widehat{\mathsf{lt}^{\mathbf{Nat}}_{S,Z}}\} \cup \Sigma, R_{\mathsf{lt}^{\mathbf{Nat}}_{S,Z}} \cup R)$
where $(\Sigma, R) = \mathcal{S}(n) \cup \mathcal{S}(S) \cup \mathcal{S}(Z)$
$\mathcal{S}(\mathsf{iterlist}(\lambda xy.C, N, l)) \doteq (\{\mathsf{lt}^{\mathbf{List}}_{C,N}, \widehat{\mathsf{lt}^{\mathbf{List}}_{C,N}}\} \cup \Sigma, R_{\mathsf{lt}^{\mathbf{List}}_{C,N}} \cup R)$
where $(\Sigma, R) = \mathcal{S}(l) \cup \mathcal{S}(C) \cup \mathcal{S}(N)$

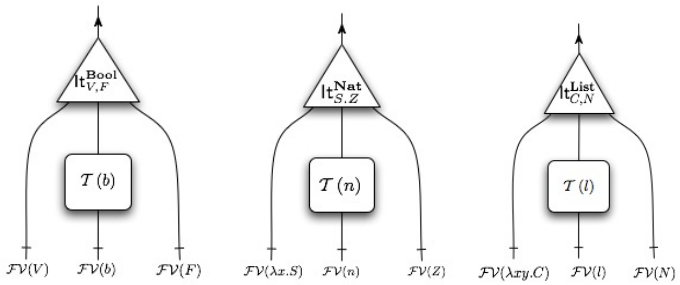
$R_{\mathsf{lt}^{\mathbf{List}}_{C,N}}$ consists of the following interaction rules (the others are similar).



Iterator symbols are introduced in pairs $(\mathsf{lt}^{\cdots}_{\ldots}, \widehat{\mathsf{lt}^{\cdots}_{\ldots}})$ where the first symbol is used for syntactic agents and the second for computation agents (similarly to $@, \widehat{@}$).

A BNL program t will be translated into a net defined in the system $(\Sigma_t, R_t) = (\Sigma_{\mathrm{tp}} \cup \Sigma_{\mathrm{BNL}} \cup \Sigma_t^0, R_{\mathrm{tp}} \cup R_{\mathrm{BNL}} \cup R_t^0)$ where $(\Sigma_{\mathrm{tp}}, R_{\mathrm{tp}})$ was defined in Section 2.

Given a BNL program t, where t is $\mathsf{iterbool}(V,F,b)$, $\mathsf{iternat}(\lambda x.S, Z, n)$ or $\mathsf{iterlist}(\lambda xy.C, N, l)$, then the net $\mathcal{T}(t)$ is given as follows.



In token-passing implementations, all terms are translated as syntax trees. Syntactic iterator agents i are turned into their computation counterparts $\widehat{i}$ by token agents. A first key aspect of our approach is that the interaction rules of the (computation) iterator agents internalise the iterator's parameters. For instance the net $\mathcal{T}\,(\mathsf{iterlist}(\lambda xy.C, N, \mathsf{cons}(h,t)))$ reduces to $\Downarrow \mathcal{T}\,(C[h/x, \mathsf{iterlist}(\lambda xy.C, N, t)/y])$.

A second key aspect is that each such new symbol will have auxiliary ports in a one-to-one correspondence with the free variables in the iterator term. We end the section with a correctness result. The proofs can be found in [1].

Proposition 1. *(***Correctness***) If t is a closed* BNL *term and z a canonical form, then: $t \Downarrow z \iff \Downarrow \mathcal{T}\,(t) \longrightarrow^* \mathcal{T}\,(z)$.*

4 Conclusions and Future Work

We have presented an approach to encoding in INs functional programs defined with recursion operators, and given the full details of the application of this approach to the token-passing implementation of a call-by-name language, which results in a very convenient visual notation for this language. The approach can be easily extended to richer sets of recursive types and other recursion operators and also to new strategies. The novel characteristics of the encoding are (i) the interaction system is generated dynamically from the program, and (ii) the internalisation of some of the parameters of the recursion operator in the interaction rules. With respect to previous work on encoding recursion in interaction nets, fixpoint operators have been studied elsewhere for interaction net implementations [4, 6], and we have shown elsewhere how a binding recursion operator (as in PCF) can be implemented in the token-passing setting [2].

A prototype system for visual functional programming has been developed, integrated in the tool INblobs [3, 10] for interaction net programming. The tool consists of an evaluator for interaction nets together with a visual editor and a compiler module that translates programs into nets. The latter module allows users to type in a functional program, visualize it, and then follow its evaluation visually step by step. The current compiler module automatically generates call-by-name or call-by-value systems. Additionally, a visual editing mode is available that allows users to construct nets corresponding to functional programs. In the current implementation there is no way to convert visual programs back to textual ones.

The token-passing translation is not however very representative of most work in this area, which has concentrated on designing *efficient* translations; [5, 7, 8] are some samples. Let $\mathcal{T}\,(\cdot)$ be one such translation. Typically $\mathcal{T}\,(t\,u)$ is constructed from $\mathcal{T}\,(t)$ and $\mathcal{T}\,(u)$ by introducing an application symbol @ with its principal port connected to the root port of $\mathcal{T}\,(t)$. Our treatment of iterators can be adapted to this setting by removing the evaluator tokens and introducing the iterator agents with the principal port immediately facing the argument. When the iterated function is a closed term, a correctness result can be easily established. Let $\lambda x.S$ be a closed term, then $\mathcal{T}\,(\mathsf{iternat}(\lambda x.S, Z, 0)) \longrightarrow \mathcal{T}\,(Z)$ and $\mathcal{T}\,(\mathsf{iternat}(\lambda x.S, Z, \mathsf{suc}(n))) \longrightarrow \mathcal{T}\,(S[\mathsf{iternat}(\lambda x.S, Z, n)/x])$

References

1. J. B. Almeida, I. Mackie, J. S. Pinto, and M. Vilaça. Encoding iterators in interaction nets. Available from `http://www.di.uminho.pt/~jmvilaca`.
2. J. B. Almeida, J. S. Pinto, and M. Vilaça. Token-passing Nets for Functional Languages. In J. Giesl, editor, *Proceedings of the 7th International Workshop on Reduction Strategies in Rewriting and Programming (WRS'07)*, volume 204 of *Electronic Notes in Theoretical Computer Science*, pages 181–198, 2007.
3. J. B. Almeida, J. S. Pinto, and M. Vilaça. A Tool for Programming with Interaction Nets. In *Proceedings of the The Eighth International Workshop on Rule-Based Programming (RULE'07)*, 2007. To appear in Elsevier ENTCS.
4. A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.
5. G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th ACM Symposium on Principles of Programming Languages (POPL'92)*, pages 15–26. ACM Press, Jan. 1992.
6. I. Mackie. The geometry of interaction machine. In *Proceedings of the 22nd ACM Symposium on Principles of Programming Languages (POPL'95)*, pages 198–208. ACM Press, January 1995.
7. I. Mackie. YALE: Yet another lambda evaluator based on interaction nets. In *Proceedings of the 3rd International Conference on Functional Programming (ICFP'98)*, pages 117–128. ACM Press, 1998.
8. I. Mackie. Efficient λ-evaluation with interaction nets. In V. van Oostrom, editor, *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA'04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 155–169. Springer-Verlag, June 2004.
9. F.-R. Sinot. Call-by-name and call-by-value as token-passing interaction nets. In P. Urzyczyn, editor, *TLCA*, volume 3461 of *Lecture Notes in Computer Science*, pages 386–400. Springer, 2005.
10. M. Vilaça. Inblobs webpage. http://haskell.di.uminho.pt/jmvilaca/INblobs/.

On Structuring Contextual Logic Programs

Salvador Abreu[1] Vitor Nogueira[1] Daniel Diaz[2]

[1] Universidade de Évora and CENTRIA, Portugal
{spa,vbn}@di.uevora.pt
[2] Université de Paris-I, France
Daniel.Diaz@univ-paris1.fr

Abstract Standardization for Prolog came during the 1990's, initially and deliberately leaving out one aspect which is essential for real world application development: the modularity mechanism. This situation has in the meantime been remedied in the current ISO proposal for modules in Prolog.
In this article we build on our previous work on Contextual Logic Programming (CxLP) and introduce mechanisms which provide much needed functionality: on one hand, a stricter specification for acceptable program structure when using contexts and, on the other, a mechanism which more effectively promotes OO-style code reuse and concealment, while retaining a lightweight syntax.

1 Introduction

Contextual Logic Programming was proposed by Monteiro and Porto [5] as a means of bringing modularity to the Prolog language. More recently, it was reintroduced as a practical extension to the Prolog language by Abreu and Diaz [1], aiming to provide a program structuring mechanism as well as fulfill some of Prolog's shortcomings when used for programming in-the-large, namely by enabling an object-oriented programming style without relinquishing the expressiveness of Logic Programs.

For their dynamically configurable structure, contexts clearly subsume the base mechanisms of most module systems, in particular the ISO/IEC Prolog Modules standard proposal [4]. This strength hides a weakness: the potentially dynamic nature of a context leads to severe difficulties in predicting its value at compile time, particularly when taking a separate compilation approach, a situation that leads to the postponement until runtime of several checks which could otherwise be performed at compile time. In section 2 we propose a mechanism which can help keep context structure in control. At the other end of the spectrum, when dealing with contexts made up of a large number of distinct units (components), when using units as (functional) building blocks for a context such as when using contexts to model ontologies, having a flat namespace for units can quickly become a hurdle. Section 3 outlines a mechanism which can overcome this limitation and still retain the notational simplicity of CxLP.

2 Context Structure

One issue with Contextual Logic Programming is the freedom it provides the programmer with, which is sometimes perceived to be excessive: the lack of compile-time knowledge about the actual runtime structure of the context has consequences over the possibility of certain types of program analysis, as it is impossible to tell which predicates will be available. Indeed, freely configurable contexts, combined with Prolog's typelessness and meta-programming practices, can create a static analyzer's nightmare. From a language design viewpoint, we wish to avoid predicate and interface declarations, in order to keep CxLP syntactically compact.

2.1 Defining the Structure of the Context

The *context search* mechanism [1] is a basic Contextual Logic Programming mechanism. It also happens to be the origin of the issues that plague Contextual Logic Programming from a static program analysis point of view: it is impossible, in the general case, to know beforehand what units will actually constitute the runtime context; therefore the available set of predicates is also unknown.

The definition of a predicate is given by the clauses in the topmost unit that defines it (i.e. the "override" semantics). This is unknown at compile time because the compilation "unit" is the CxLP *unit*, not the *context*. The actual runtime context in which the present unit will occur may include an arbitrary set of units with the correspondingly arbitrary set of predicate definitions.

A workaround for this problem was proposed in [1] by means of the `:>>` operator which allows for a particular unit to specify a predicate (`context/1`) which will dictate what actual context it will be used, at runtime.

In pursuing simplicity and compactness, we relocate this notion of structure into the unit directive itself. Instead of resorting to the `context/1` hack, the directive simply specifies a prefix[1] of the context in which the unit can appear.

As an illustration, consider the units `person(NAME, ADDRESS, BIRTHDATE)` and `student(ID, PROG)` to represent the concepts that a *student* "is-a" *person*, meaning it should inherit everything a person has whilst extending it with new attributes (in this case, an ID number and a study program code.) Using the extended directive to express such concepts we'll have the following unit declarations:

```
:- unit person(NAME, ADDRESS, BIRTHDATE).
:- unit student(ID, PROG).person(NAME, ADDRESS, BIRTHDATE).
```

Notice that this notation is backward compatible, i.e. the declaration for unit `person/3` looks the same as previously.

We explicitly state the prefixes of the contexts in which the goals of these units can occur: any goal of unit `student/2` is evaluated knowing that the current context starts with `[student(_, _), person(_, _, _), ...]`.

[1] The prefix starts with the current unit, looking at the context as a stack, headed by its topmost element.

Finally, this declaration allow us to have direct access to the unit parameters of unit `person/3` from inside unit `student/2`. The justification for this programming technique is simple, from an OOP point of view: proper inheritance may require sharing of the "superclass" instance variables.

3 Scoped Units

Most modern programming languages provide layered modularity mechanisms in which visibility is controlled. Take for example Java packages or XML namespaces. In the case of CxLP, units are a program-structuring component which is combined to form contexts. The simple CxLP specification does not effectively account for accidental naming collisions, such as would happen with unit names in situations where there are many of them.

To address this situation, we introduce a new mechanism: *local units*, for which the unit name/set of clauses binding is only valid from within an other, specific unit, called the "interface unit." Moreover, the context search procedure described in [1] has been modified to ignore local units, unless they are topmost in the context: this effectively makes predicates defined in local units invisible to all but explicit accesses, which can only be originated in the interface unit.

An example of a program which uses local units:

```
:- unit lists.
rev(L1, L2) :- utils :> rev_aux(L1, [], L2).
...

:- unit lists/utils.
rev_aux([], L, L).
...
```

The second unit declaration (`lists/utils`) is actually defining a unit called `utils/0` which is only visible directly from within unit `lists/0`. Therefore there may be more than one local unit with this same name, there being no conflict as it will be local to the interface unit. Interface units still share a single, flat name space.

4 Concluding Remarks

One issue which has to be satisfactorily resolved in order for a system based on GNU Prolog/CX to scale well is the structuring of the unit namespace. We are currently working on possible approaches to dealing with this problem, some of which look very promising and we expect to be able to report on that shortly.

We are currently evolving the techniques presented in this article to include some of them in the ISCO [2] compiler. It is in our plans to do extensive benchmarking of various code generation strategies, some of which will include static contexts.

As the scope of ISCO applications keeps growing, these techniques should prove more important as they should enable the development of tools which can establish properties of programs developed with GNU Prolog/CX [1]. This will mean the development of global analysis tools along the lines of those available in Ciao Prolog [3], e.g. by making use of abstract interpretation. These tools will serve optimization, debugging and documentation purposes.

Now that we have a working prototype implementation, we plan to compare it with other approaches to the modularity problem. This will be the subject of a future article.

References

1. Salvador Abreu and Daniel Diaz. Objective: in Minimum Context. In Catuscia Palamidessi, editor, *Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings*, volume 2916 of *Lecture Notes in Computer Science*, pages 128–147. Springer-Verlag, 2003. ISBN 3-540-20642-6.
2. Salvador Abreu and Vitor Nogueira. Using a Logic Programming Language with Persistence and Contexts. In Osamu Takata, Masanobu Umeda, Isao Nagasawa, Naoyuki Tamura, Armin Wolf, and Gunnar Schrader, editors, *Declarative Programming for Knowledge Management, 16th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2005, Fukuoka, Japan, October 22-24, 2005. Revised Selected Papers.*, volume 4369 of *Lecture Notes in Computer Science*, pages 38–47. Springer, 2006.
3. Daniel Cabeza and Manuel Hermenegildo. The Ciao Module System: A New Module System for Prolog. In Ins Dutra, Vtor Santos Costa, Gopal Gupta, Enrico Pontelli, Manuel Carro, and Peter Kacsuk, editors, *Electronic Notes in Theoretical Computer Science*, volume 30. Elsevier Science Publishers, 2000.
4. ISO/IEC JTC1/SC22/WG17. Information technology – Programming languages – Prolog – Part 2: Modules. Technical Report DIS 13211, ISO, 2000.
5. L. Monteiro and A Porto. Contextual logic programming. In Giorgio Levi and Maurizio Martelli, editors, *Proceedings of the Sixth International Conference on Logic Programming*, pages 284–299, Lisbon, 1989. The MIT Press.

Sessão 2B: Computação Móvel e Ubíqua

RFID based Monitoring and Access Control System

Filipe Lourenço and Carlos Almeida

DEEC, AC-Computadores.
Instituto Superior Técnico. Technical University of Lisbon.
Av. Rovisco Pais, 1. 1049-001 Lisboa. Portugal.

Abstract. This paper addresses the design and implementation of a monitoring and access control system based on RFID (Radio Frequency Identification) wireless technology. The system, that allows the identification and authentication of people and objects, controlling their access to restricted areas (e.g. laboratories), has a hierarchical structure being composed of four parts: Terminal, Server, Camera and Alert Device (e.g. PDA (Personal Digital Assistant)). When the Terminal detects the transponders (RFID tags), it captures a photo and sends all this information to the Server, which records the event and queries a database to evaluate access permission. There are several parameters of authentication, including: job post, timetable, and type of material. The system also enables ordering and returning of material from different laboratories. When illicit acts or abuses are detected, a warning is transmitted to a surveillance post (Alert Device).

1 Introduction and Motivation

In a modern world, with little time for bureaucracy, control over property is a major factor of concern to any institution or company. All companies have restricted access areas or places where there must be strict control on the mobilization of material to prevent its disappearance (e.g. libraries, stockrooms, etc.). A solution for easy access by authorized personnel and access to material that saves time that would otherwise be wasted by searching store keys and/or waiting for the security to arrive should be welcome. Such a solution would also improve ease of returns or requests of equipment. Furthermore, it allows a thorough inventory on all the material, thus, avoiding mysterious disappearances.

The goal of this project was to develop a monitoring and surveillance system, based on a hierarchical structure and using wireless technology, to control access to restricted areas. The system should operate in a truly continuous fashion and be able to detect and record all events (with a photo) that occur in the restricted area. The system should also allow the identification of people and objects. People may have one or more associated job posts. Each job post has one or more associated schedules and multiple permissions to request material. The schedules may be associated to one room, more than one or all the rooms of the system. For each access where material is checked out, the system should

update the stock of available material. When material is checked into the room (any room belonging to the system) the system should update the stock of available material and its current location. If problems or illicit acts are detected they should be registered and transmitted to an alert device.

2 Overview of Related Technologies

Due to significant technology advances and cost reductions, embedded systems are growing in the market in various areas, such as: portable MP3 players, Personal Digital Assistants (PDA), cellular phones, electronic gadgets and sensors networks. Various types of wireless communications protocols have also emerged in the last years. This contributed to a new era of mobility of embedded systems. The Bluetooth [1] and Wi-Fi [2] exploded quickly in the market as a consequence of their ease in forming ad-hoc networks that use the devices (e.g. laptops, mobile phones, game consoles, etc.). However, the high energy consumption prevents its use on systems that depend on a small battery. The ZigBee (IEEE 802.15.4 protocol) [3] improved efficiency of radio-frequency communications using very low power consumption. This is possible because the sensors using it are dormant most of the time, and it is ideal for applications requiring low data transfer and long life batteries.

With the increasing potential of embedded systems along with size reduction, the new video surveillance systems have gained much ground in the market, having released many of the old security posts (not only responsible for the premise's security, but also for room's access by people, etc.). The surveillance guard is often far from the place where the images are captured, but given the large number of sensors developed in recent times, not all images need to be observed in real time. Technological advances allow a small number of personnel to watch images of thousands of cameras spread over several venues, through an image analysis system that only shows the images captured in real time to detect unusual behavior (such as suspicious attitudes, sudden movement, the abandonment of objects). When illicit acts are detected, a message is transmitted to a team in the place, or nearby, to take preventive measures.

2.1 RFID Technology

RFID (Radio Frequency Identification) is a technology that allows the identification of a transponder or the transfer of data to a terminal through the use of radio waves [4]. Announced as the replacement for traditional barcodes, RFID's wireless identification capabilities promise to revolutionize our industrial, commercial, and medical experiences. The main thing about this utility is that it makes gathering information about physical objects easy. Information of transponders can be transmitted to multiple objects simultaneously, through physical barriers, and from a distance. A RFID system consists of terminal (also called reader or interrogator) and transponders (or RFID tags). The terminal

communicates with the transponders within its wireless range to collect information. The RFID systems can be classified into three distinct categories: passive, semi-passive and active [4].

Passive RFID is of interest because the transponders do not require batteries or maintenance. All the power needed for broadcast is provided by the terminal. The transponders also have an indefinite operational life and are small enough to fit into a practical adhesive label. The cost of manufacture is very cheap. The maximum range of passive RFID transponders is approximately three meters [5].

Active transponders require a power source. They are either connected to a powered infrastructure or use energy stored in an integrated battery. Transponder's lifetime is limited by the stored energy, balanced against the number of read operations the device must undergo. The price of manufacture is very expensive compared to the passive transponders. The maximum range of active RFID transponders is approximately one hundred meters.

Semi-passive transponders use internal batteries to power their circuits, but they also rely on the terminal to supply the power for broadcasting information. Apart from the battery, they work like passive transponders. The price of manufacture is higher than passive transponders and the maximum range is approximately three meters.

Passive RFID transponders Passive RFID transponders, without a power supply of their own, depend upon the electromagnetic field of the terminal [6]. The coupled energy is rectified and the voltage multiplied to power up internal circuits. Passive RFID transponders communicate with the terminal by inductive coupling [5]. This is a simple mechanism based on Faraday's principle of magnetic induction (see Figure 1) [7]. A current flowing through the coil of a reader produces a magnetic field. This field causes the transponder coil in its vicinity to generate a current. Communications between the terminal and the transponder are through a mechanism called load modulation. Any variation of the current in the transponder coil causes a small current variation in the terminal coil due to the mutual inductance between the two. This variation is detected by the terminal. A transponder varies the current by changing the load on its coil antenna, and hence the mechanism is called load modulation. Because of its simplicity, inductive coupling was initially adopted for passive RFID systems.

With this mechanism the maximum range of passive RFID transponders is approximately three meters. The transponders may have various shapes and sizes, depending on its purpose (for example, some transponders are the size of a grain of rice and others are thin enough to fit into a practical adhesive label). Each transponder contains an encoding and decoding circuit, communications control, antenna and memory.

This technology is ideal for tracking people, items and equipment in real time. The transponders do not require a line of sight with the terminal because all the information is broadcasted via radio. The transponder can stand a harsh environment and does not need power supply. Also the reading range is very

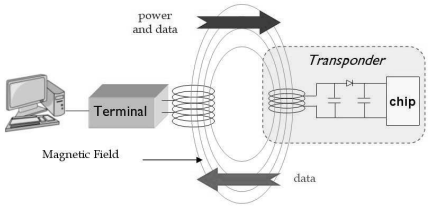


Fig. 1. Example of near-field communication using inductive coupling.

good, and, due to anti-collision mechanisms, the terminal can read or write in multiple transponders simultaneously.

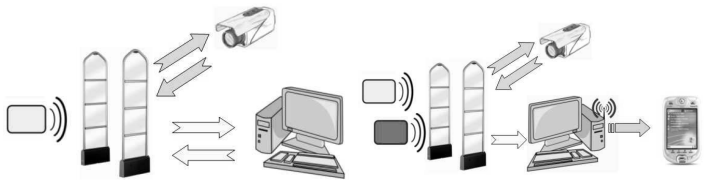
RFID currently presents on the market several types of applications: for identification and authentication of users (access control); for the recognition of objects (storage and logistics); and for the detection of objects (alarm systems and anti-theft systems) [5, 7–9]. One example of a success application is the RFID Library [10]. In this library, all the books have one transponder that helps to find them in the shelf and prevents from stealing, with the anti-theft alarm. With RFID transponders in each book it is quicker to find the book, to request/return, and to find its correct spot.

3 Application Scenario

The infrastructure addressed in this project consists on: a RFID terminal, a camera, a server (computer) and a PDA. It is possible to implement this infrastructure in one or more buildings with several types of rooms (for example, laboratories, class rooms, storages, etc.). The access to each room should have at least one RFID terminal and one camera. Alternatively, the camera may be shared by some access terminals. When the terminal detects at least one transponder, the authentication process begins. During this process, all the UIDs (unique identifications) from the detected transponders are collected and stored in local memory. The terminal also instructs the camera to capture a photo. After a short period of time, all the data is transmitted to the server by a message, containing all the UIDs and the photo captured. The server collects all the data and makes a historical log. It, then, proceeds to analyzing the request and verifying if all the authentications are fulfilled. The server replies to the terminal with the results of the authentication request. There are two types of results: access allowed or access denied. In the former case, the terminal opens the doors and allows the access to the user and his belongings. After a short period, the doors will close. In the latter case, the access is denied and a short message stating the reason of the refusal is displayed. This is the end of the normal authentication process. A simple scheme of this process is represented in the Figure 2(a) (the doors of the terminal are not represented on the Figure).

When the access is allowed the doors of the terminal are opened for the user to pass. During this period of time the system is vulnerable because people

or objects may leave the room without the authentication request. Although, the authentication process is not properly executed, all the new transponders detected while the doors are open are collected and stored in local memory. A picture is also taken and stored in memory. When the terminal closes the doors, all the data is transmitted to the server. The server collects all the data and makes a historical log. Then sends a message to the alert device (in this case, to the PDA) in order to alert the security. The message contains information about the user, the material, the place where occurred and the picture captured (see Figure 2(b)). Any other problem detected by the system (e.g. failure in the camera) is also transmitted to the alert device in order to solve it in short time.



(a) Normal process. (b) Illicit act or problem detected.

Fig. 2. Examples of authentication processes.

4 System Architecture and Components

The Monitoring System was developed based on a hierarchical structure, and is composed by four individual components: Server, Terminal, Camera and Alert Device (see Figure 3) [11]. Each component was independently developed and programmed in C#.

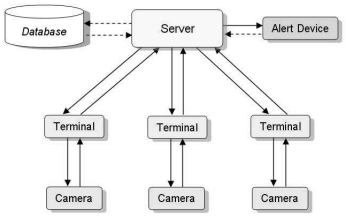


Fig. 3. System architecture and components.

4.1 Server

The server acts as the central storage system and as the validation data center. All the information (historical log, access rules, users, rooms and so on) is stored on a database (that was designed in SQL). The server accepts requests from two devices (terminal and alert device), but it is possible to configure it to accept other requests, if desired.

The requests made from the terminals are those for the room access or to report a problem or an illicit act. They all are received, logged and checked. The access requests are checked with the database and then the server replies to the terminal with a message, allowing or denying the access to the room. The authentication process consists of several steps. The first one is to check if the UID belongs to people or objects. Only one person is allowed for each access. However, it is possible to request or return more than one piece of material. It is necessary to validate the permissions for: checking out material, room's access and the schedule of access. All the UIDs, from the detected transponders, that are not recognized by the system are ignored. These are usually from people's belongings (e.g. books) or from other authentications systems (e.g. transport tickets). When the requests involve material the stocks are updated. Finally, a message is sent to the terminal to allow or deny the access.

The server may also receive a message reporting an illicit act. All the information is logged on the database, and the server sends a message to the alert device reporting the act. The server also sends messages to the alert device whenever a system problem is detected. For example, if the authentication request has no picture, it means that the camera is down, and maintenance is required.

The alert device only sends messages to the server for: request new illicit acts, requesting pictures from the illicit acts and to notify the server that the problem is solved or being take care of.

The communication between the terminal and the server is provided by a TCP/IP (Transmission Control Protocol / Internet Protocol) connection [12]. The Server has a fix IP address and accepts requests from all system terminals. To allow the communications between the alert device and the server, an application using Web Services [13] was developed.

The server interface, in addition to system configuration (user and equipment registration, rules definition, etc.), allows to monitoring the restricted area in real time, such as the occupancy of each room, the list of all users in the area, check-in data and so on. It is also possible to present some statistics, like the occupation by room, ranking by frequent users, occupation by date, and so on. In Figure 4 some system interface panels are shown.

4.2 Terminal

The terminal is responsible for establishing the physical separation between the public area and the restricted area, detecting the transponders and making authentication requests to the server. It is in standby mode, being activated whenever a transponder is detected. When that happens, it sends a request to the

(a) Historical Log. (b) Access rules.

(c) Register Users. (d) Monitoring.

Fig. 4. Examples of several panels of the server interface.

camera to capture a photo. All UIDs from detected transponders (with duplicates eliminated – a given transponder can be read more than once) are collected and, together with the photo, are sent in a message to the server. This process should take a maximum of two seconds.

Once the terminal receives the reply from the server, it acts accordingly: if the access is denied, the doors stay closed and a message is presented to the user explaining why the access is not possible; if the access is allowed, the doors will open and a message greeting the user is presented. In both cases, after a short period of time, the terminal goes back to standby mode (closing the doors if necessary) and cleaning up the information collected.

While the doors are open the antennas of the terminal are active and detecting new transponders. If a new UID is detected, it means that something or someone is entering or leaving the room without a proper authorization. This is considered an illicit act and the terminal lights up the red LED or sounds a buzzer. The terminal does not know if the transponder belongs to a person or an object. This information is only accessible by the server. Consequently, after all the new UIDs from transponders are detected and the doors closed, a

message will be sent to the server. The message is composed by all the new UIDs detected, the UID from the user (that was transmitted in the first reply from the server) and a new photo (that is captured as soon as the new UID's transponder is detected). After the message is sent, the terminal cleans up the memory stack and it is ready for the next request.

If the camera is down, the system still works and sends the same messages to the server. This happens because the system is hierarchical and the lack of one device does not compromise the whole system. However, if the server is down, for security reasons the admittances are not allowed. For other unexpected problems there is a Watchdog to ensure the proper functioning of the terminal.

The access terminal needs to ensure that only authorized people can access the restricted areas and that only some of those can request material (e.g. avoiding the movement of small size equipment over the terminal). These requirements were not fulfilled by any RFID terminal in the market, hence the requirement to design one (see Figure 5). The ideal solution for this kind of application would be a cabin. But the large dimensions and the limited space inside (for the access of larger material, e.g. ladders for maintenance) make this an unpractical solution.

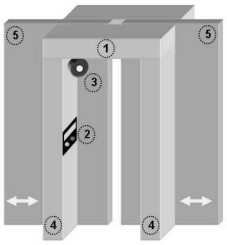


Fig. 5. Suggested terminal designed for this application. Legend: 1 - RFID reader; 2 - LCD panel; 3 - camera; 4 - antenna; 5 - sliding doors.

4.3 Camera

The camera is used for capturing photos of the admittances. It may take pictures of every access (or request) or only of detected illicit acts. This device may be shared by several terminals. A photo is taken upon a request by a terminal, which receives the reply message. The message contains the photo as an array of bytes. A conversion to JPEG is latter performed by the server.

4.4 Alert Device

The alert device is used to alert the security when the system requires human intervention. The device used as an alert device was a PDA which does not restrict the mobility of the security.

There are two types of messages which require human intervention: failure in a system device (e.g. camera out of order) and detection of illicit acts. These messages are simple and are presented in the PDA as soon as they are received. In case of a detection of an illicit act, it is possible to request from the server the photo that was captured. To avoid waste traffic, the photo is only transmitted when requested. After the security takes care of the issue, it is possible to notify the server to store the alert and to not transmit it anymore to the alert device.

In the current implementation the alert device is connected to a WLAN (Wireless Local Area Network) and communicates with the server by polling. To optimize this solution, it is possible to notify the alert device by a hired mobile service such as SMS (Short Message Service) or MMS (Multimedia Messaging Service). After the alert device is notified by the SMS, it connects to the server and collects all the new information. It is also possible to add or replace the PDA for another mobile or fixed device.

5 Improving System Security

One of the biggest issues with authentication systems is the risk of fake identification [14]. This is also a risk in passive RFID systems. There are two possible ways to try to fake the identification: eavesdropping communication and cloning the transponders [15, 16]. To increase the security of the system developed, three mechanisms were designed: exclusivity, entries count, and password. The goal of these mechanisms is to try to detect cloned transponders. These mechanisms may imply the ability to write on the transponders when accessing the system.

5.1 Exclusivity

The first mechanism developed, named *exclusivity*, has a very simple and basic concept: only allows one transponder with the same identification (UID) checked in simultaneously on the system. This means that no further access can be made to the system, with this transponder, except from the inside room where it is signed in. Any other request will not be accepted by the server and will deactivate the transponder. If the transponder is deactivated and the user in the room requests to sign out, the system will allow but will not accept more requests from that transponder.

5.2 Entries Count

The second mechanism developed, named *entries count*, complements the first one. This mechanism increments and writes the number of entries on the transponder each time there is a sign in into the system. Each transponder has an array with the total number of entries in each room (modulo 99).

When the transponder makes a request to the system, the terminal reads the total of entries that the transponder made in that specific room, from the entries count array. Then, this number is compared with the number of total accesses

made by this transponder that are registered on the system. If the number of entries in the system is higher than 100, only the two last digits are compared. If the two numbers match, everything is all right. Otherwise, it means that the transponder is cloned and the transponder is deactivated. This method prevents the use of two transponders (with the same identification) to use the same room not simultaneously.

A more efficient and secure way would be to check the whole entries array with the system. However, it would be easier to get all the information from the transponder by eavesdropping communication.

If the system has a large number of rooms, the internal memory of the card might not be enough. In this case, there are two distinct options: buy cards with more memory or optimize the memory. The second option consists in the aggregation of entries of some rooms for identification instead of single entry count room identification.

5.3 Password

The third mechanism developed, which can be an alternative to the *entries count*, was named *password*. Each time the transponder signs in the system, the password is checked with the one that is stored in the server for this transponder. If the password does not match, the transponder is deactivated. Otherwise, the password matches and is discarded. A new password is generated, for the next check in and is stored in the server and is also stored on the transponder.

This mechanism works like a new and different key for each sign in access. It also prevents the use of cloned transponders since each key is only valid for one access: the next one. In the Figure 6 it is possible to see a simple example of how this mechanism works.

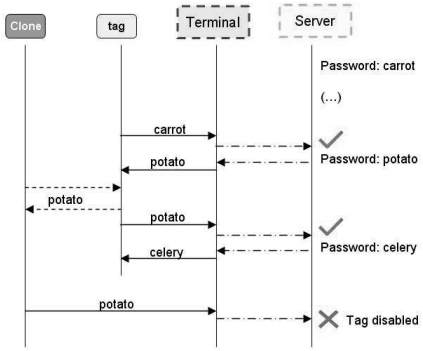


Fig. 6. Example of the password mechanism. First, the password is *carrot*. After the authentication the password changes to *potato*. Before the next sign in, the transponder is cloned. The original transponder signs in again and the password is changed to *celery*. The cloned transponder will be detected as soon as it tries to sign in the system.

The three developed mechanisms can be used to increase the security of the system.

6 System Performance

The system performance was tested with a RFID kit with an antenna with 10 cm of range [17]. The transponders were accurately detected by the terminal and all requests were analyzed correctly (with one or several transponders). Also all the illicit acts committed were correctly captured by the camera and transmitted with success to the alert device.

In order to determine the size of the messages sent by the terminal to the server, the minimum and maximum size of information (in bytes) contained in the transponders and in the photos were analyzed (see Table 1).

Table 1. Minimum and maximum message size between terminal and server.

	Items		Information (byte)	
	Min.	Max.	Min.	Max.
Photo	1	1	5960	18 630
Transponder	1	99	14	1386

With this information it is possible to estimate the medium size of the message sent by the terminal to the server. In a typical use it is expected that each user carries between one and ten transponders that are detected by the terminal. The size of the message will be, in most cases, less than 20 KB, which is a very acceptable size for the server (if needed old information can be moved or rewritten).

In order to test the three security methods developed, a third terminal application was implemented. This application has an independent database and also reads the transponders identification. Before sending the message to the server, this special application is able to mask one or more transponders identification in order to fake the transponders UIDs. The masking process allows the simulation of cloning transponders by the manipulation of data in order to test each one of the security methods developed. The system was able to detect with success the cloned transponders, denying the access to the room after detecting more than one transponder with the same identification.

7 Conclusions and Future Work

The results show that this system simplifies the bureaucratic methods and successfully automates the access to a restricted area, thus increasing the autonomy of users in the enclosure, in this case, students. Due to rapid technological advances in the area of RFID, the vicinity cards are now entering the market

presenting a good alternative to contactless smart cards. This was made possible due to higher transmitted signal strength, which allows for greater range of communication. After completion of the project, several adjustments were made to the application developed that enabled it to use both types of cards (contactless smart cards, ISO 14443 Type-A and ISO 14443 Type-B, and vicinity cards, ISO 15693) as a method of authentication, thus presenting a ready application to current and future needs.

In the future, this work may have more applications in the areas of home automation and security. For home automation it can have interactions with other systems of control and monitoring, optimizing energy consumption, such as ventilation system size as a function of room occupancy, turn off monitors and lights after all users leave, switching on lights, if necessary, when a user enters, just to name a few. For security purposes, it can be implemented with facial recognition systems as well as biometric readings to reduce the risk of cloned cards. Another improvement is server replication to increase availability.

References

1. McDermott-Wells, P. In: What is Bluetooth? IEEE Potentials (Dec. 2004) 33–35
2. Wi-Fi Alliance: Wi-Fi Alliance Home Page. http://www.wi-fi.org
3. Heile, B., Adams, J. In: Busy as a ZigBee. IEEE Spectrum (Oct. 2006)
4. Want, R. In: An Introduction to RFID Technology. Volume 5. Pervasive Computing (Jan. 2006) 25–33
5. Finkenzeller, K. In: RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification (2nd Ed.). J. Wiley and Sons (2003) 161–228
6. Chawla, V., Ha, D.S. In: An Overview of Passive RFID. Volume 45. IEEE (Sept. 2007) 11–17
7. Ahson, S., Ilyas, M. In: RFID Handbook: Applications, Technology, Security, and Privacy. CRC Press (2008) 125–278
8. Jones, E.C., Chung, C.A.: RFID in Logistics: A Practical Introduction. CRC Press (2007)
9. Weinstein, R. In: RFID: A Technical Overview and Its Application to the Enterprise. Volume 7. IT Professional (May 2005) 27–33
10. BookTec Information Co.: LibBest - Library RFID Management System. http://www.rfid-library.com (2008)
11. Lourenço, F.M.: Sistema de Monitorização com Estrutura Hierárquica para o Acesso a Áreas Restritas. Master's thesis, Instituto Superior Técnico - Universidade Técnica de Lisboa, DEEC (Abril 2009)
12. Kurose, J.F., Ross, K.W. In: Computer Networking: A Top-down Approach Featuring the Internet - Third Edition. Addison-Wesley (2005)
13. Cerami, E. In: Web Services Essentials. O'Reilly (2002) 3–67
14. Thornton, F., Haines, B., Das, A., Bhargava, H., Campbell, A., Kleinschmidt, J. In: RFID Security. Syngress (2006) 29–54
15. Jules, A. In: RFID Security and Privacy: A Research Survey. Volume 24. IEEE Journal on Selected Areas in Communication (Feb. 2006) 381–394
16. Rieback, M., Crispo, B., Tanenbaum, A. In: Is Your Cat Infected with a Computer Virus? Fourth Annual IEEE International Conference on Pervasive Computing and Communications (March 2006) 169–179
17. : EV-100 Evaluation Board Manual v2.0. 3ALogics Inc. (Oct. 2005)

PIPE: Uma infra-estrutura genérica de serviços para ambientes de computação ubíqua

Bruno Félix, Nuno Preguiça

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

Resumo Nas últimas décadas tem-se assistido a um crescimento significativo do número e poder dos diferentes dispositivos computacionais. Este crescimento traduz-se na coexistência de duas grandes classes de equipamentos. Por um lado aqueles que permeiam de forma quase ubíqua os espaços que frequentamos e por outro lado, um grande número de equipamentos móveis que acompanham no utilizadores no seu dia-a-dia. No entanto, estas duas classes de dispositivos têm existido em mundos largamente separados. Neste artigo apresenta-se o PIPE, um sistema ubíquo para espaços públicos que permite a utilizadores equipados com dispositivos móveis usufruir dos recursos (e.g., ecrãs partilhados, capacidade computacional) fornecidos pelo sistema, podendo correr um conjunto de aplicações pré-definidas, ou as suas próprias aplicações na infra-estrutura, estendendo a capacidade dos dispositivos portáteis.

Key words: computação ubíqua, dispositivos móveis, ecrãs partilhados, execução remota.

Resumo In the last decades we have witnessed a significant growth in the number and power of computing devices. These can be divided in two broad categories. First, the devices that pervade most of the spaces we use, and second the mobile equipments carried by the users in their day-to-day life. However, up until recently both these classes of devices coexisted without much interaction. In this document, we present PIPE, an infrastructure aimed at public spaces that allows users equipped with mobile devices, to take advantage of the provided resources (e.g. shared displays, computing power), allowing them to run existing, or their own distributed applications, in machines that are part of the infrastructure.

Key words: pervasive computing, mobile devices, shared displays, cyber foraging.

1 Introdução

A crescente ubiquidade dos dispositivos computacionais é uma realidade que tem vindo a alterar profundamente o nosso modo de vida. Estes têm-se tornado

cada vez mais pequenos, potentes e baratos, sendo por isso bastante comum encontrar espaços onde coexistem dispositivos pertencentes à infra-estrutura (computadores, ecrãs, projectores, etc.) e dispositivos pessoais dos utilizadores (telemóveis, PDAs, etc) [1].

Até ao momento estas duas classes de dispositivos têm estado largamente dissociadas. No entanto, a sua combinação pode abrir oportunidades interessantes para criar novas aplicações e serviços. Estas aplicações, podem usar os dispositivos móveis como meios privilegiados para aceder aos recursos que existam num determinado espaço e para manter informação pessoal do utilizador, ao mesmo tempo que tiram partido do poder de processamento, resolução e largura de banda que os dispositivos instalados na infra-estrutura oferecem.

A união destas duas classes de dispositivos levanta um conjunto de novos desafios importantes [11,8]. Entre estes, encontra-se a necessidade de combinar um conjunto diversificado de meios computacionais, que podem estar distribuídos por diferentes dispositivos de forma a fornecer a informação desejada ao utilizador final. Este problema consiste não só na definição da arquitectura que permite compor aplicações a partir dum conjunto distribuído de recursos, mas também na forma como estes serviços podem ser instanciados e migrados de forma a executarem nos dispositivos mais apropriados. Por exemplo, uma aplicação que apresente ao utilizador informação obtida a partir de múltiplas fontes web pode ter um módulo a executar na infra-estrutura para executar este processamento, usufruindo de uma melhor conectividade e capacidade de processamento.

Neste contexto, este artigo propõe uma plataforma que permita a criação de aplicações que utilizam os recursos disponíveis na infra-estrutura para fornecer um melhor serviço aos utilizadores finais. Este sistema, designado de PIPE (Pervasive Infrastructure for Public Environments), permite que um utilizador munido apenas do seu dispositivo móvel possa tirar partido dos recursos computacionais disponibilizados pelas várias máquinas que compõem esta infra-estrutura.

No sistema PIPE executa um conjunto de serviços e aplicações. Um serviço disponibiliza uma funcionalidade específica que pode ser utilizada apenas por outros serviços ou aplicações. As aplicações fornecem um serviço ao utilizador final. Enquanto os serviços executam apenas nos nós da infra-estrutura, as aplicações podem também executar nos dispositivos dos utilizadores. Quer as aplicações quer os serviços podem ser instaladas dinamicamente nos dispositivos apropriados.

A nossa visão é que este sistema possa executar num espaço público (e.g. um centro comercial), onde existem à partida uma grande quantidade de dispositivos computacionais espalhados pela infra-estrutura e um número substancial de utilizadores. O sistema incluirá um conjunto pré-definido de aplicações e serviços disponibilizados pelo espaço público em que executam. Os utilizadores podem aceder a estes serviços, fazendo o carregamento dinâmico das aplicações nos seus dispositivos se necessário.

Ao contrário de outras soluções para este tipo de ambiente [7,6,10], a nossa solução permite ainda carregar dinamicamente aplicações do utilizador na infra-

estrutura. Esta funcionalidade possibilita uma maior flexibilidade, não ficando o utilizador limitado a usar as aplicações disponibilizadas pelo espaço público, mas podendo usar as suas próprias aplicações.

O resto deste documento está estruturado da seguinte forma. A secção 2 descreve o trabalho relacionado. A secção 3 discute brevemente os requisitos relevantes para um sistema desta natureza. A secção 4 apresenta o desenho do sistema PIPE. Na secção 5 descrevem-se duas aplicações que usam o sistema PIPE, seguindo-se a avaliação do sistema e algumas conclusões finais nas secções 6 e 7 respectivamente.

2 Trabalho relacionado

Os avanços tecnológicos permitiram a criação de sistema de computação ubíqua [14,11]. Estes sistemas exibem uma grande variabilidade, sendo possível identificar duas categorias especialmente relevantes para o tipo de plataforma apresentada neste artigo. Por um lado, aplicações específicas que tiram partido de computadores presentes no espaço em conjunto com os dispositivos móveis disponíveis. Por outro lado, plataformas ubíquas para o fornecimento de serviços, que permitem aos utilizadores aceder a um conjunto mais vasto de recursos e aplicações.

Na primeira categoria podemos englobar diferentes trabalhos - e.g. [13,5]. Um exemplo representativo é o sistema Bluscreen [5], que permite mostrar publicidade, ou outros conteúdos informativos em ecrãs públicos. Estes conteúdos são escolhidos tendo em conta os dispositivos bluetooth detectados nas proximidades. Este sistema não requer software específico por parte dos utilizadores, tendo em consideração nos conteúdos mostrados o facto dos utilizador já os terem observado anteriormente ou não. Assim, este sistema funciona em situações em que à partida não se conhecem os interesses e tipologia dos utilizadores. O sistema PIPE permite criar uma solução semelhante.

A segunda categoria engloba sistema ubíquos mais complexos e com diferentes objectivos. O Celadon [7] é um exemplo deste tipo de plataformas, que visa criar uma infra-estrutura ubíqua que permita aos dispositivos móveis presentes num dado espaço tirar partido dos recursos e serviços que a infra-estrutura fornece. Os serviços disponibilizados encontram-se agregados em zonas, que correspondem a espaços concretos (e.g. estações de metro, centros comerciais, etc).

O AlfredO[10] apresenta uma arquitectura orientada aos serviços, para suportar o controlo e execução de serviços fornecidos pela infra-estrutura a partir do dispositivo pessoal do utilizador. Os serviços são decompostos em três camadas (dados, lógica e visualização), podendo uma ou várias destas camadas ser transferidas e executadas no cliente móvel, oferecendo um bom grau de flexibilidade.

O PIPE foi influenciado pela abordagem seguida por estes dois sistemas, na medida em que oferece uma solução baseada em aplicações disponibilizadas pelo sistema, que podem ser consumidas pelos utilizadores. No entanto, permite também que estes usem a infra-estrutura para instalar e correr as suas próprias aplicações.

Recentemente, vários trabalhos exploraram a possibilidade dos utilizadores de computadores portáteis usarem os recursos disponíveis na infra-estrutura para executarem computações arbitrárias - e.g., [3,12].

O Slingshot [12] permite a execução de código arbitrário em máquinas de um *hot-spot*, através da replicação de máquinas virtuais em que correm as aplicações dos utilizadores. Essas réplicas são obtidas a partir de um servidor, bem conhecido pelo utilizador, que actua como réplica primária e é usado como repositório "confiável"do estado da aplicação. A solução proposta neste artigo é mais leve, e baseada na execução de programas Java. Adicionalmente, ao contrário do Slingshot, as novas aplicações ficam integradas no ambiente de computação ubíqua, podendo aceder aos serviços disponibilizados no sistema.

3 Requisitos funcionais

Nesta secção iremos abordar brevemente alguns dos requisitos funcionais que identificámos e que foram determinantes em algumas escolhas que fizemos aquando do processo de desenho da nossa solução.

Custo: A implementação de uma arquitectura para o fornecimento de serviços ubíquos não deve comportar um custo muito elevado. De preferência deve poder tirar partido de muito do hardware que já existe espalhado pelos diferentes locais, e também não deve exigir equipamento específico da parte dos clientes. Mesmo nestas condições, uma infra-estrutura deste tipo terá alguns custos. O estudo dos modelos de negócio associados à disponibilização deste tipo de infra-estrutura está fora do âmbito deste trabalho, mas poderia passar pelo suporte pelo espaço público como forma de atracção dos cliente, usando receitas de publicidade apresentada na infra-estrutura ou usando um modelo de partilha de custos com os utilizadores.

Flexibilidade: Uma plataforma desta natureza deve não só contemplar os mecanismos para fazer uma gestão fácil dos serviços que a infra-estrutura oferece, mas também permitir ao utilizador executar as suas próprias aplicações ou serviços.

Tempo de resposta: Tendo em conta que esta plataforma tem um modelo que é mais facilmente aplicável a espaços públicos, como por exemplo centros comerciais, a questão do tempo de resposta é bastante importante. Neste tipo de ambiente, não é expectável que o utilizador tenha disposição para esperar bastante tempo para interagir com a infra-estrutura, pelo que esse facto deverá ser considerado.

Segurança e auditabilidade: Tendo em conta que há a possibilidade dos utilizadores correrem as suas próprias aplicações a questão da segurança torna-se particularmente premente. Neste contexto existem vários problemas distintos, dos quais se destacam os seguintes.

Primeiro, a protecção da infra-estrutura face à execução de software arbitrário. Este foi o único problema tratado de forma completa no nosso sistema, usando um modelo de *sandbox*, em que as aplicações executam de forma isolada recorrendo aos mecanismos de segurança do sistema Java.

Segundo, o controlo de acessos de uma aplicação relativamente aos serviços que pode utilizar. Para lidar com este problema seria possível combinar a utilização do sistema *Kerberos* com um sistema de *capacidades*, em que um cliente apresentasse aos vários serviços um *ticket* com as suas permissões.

Finalmente, é interessante providenciar mecanismos que permitam certificar ou auditar os resultados das computações efectuadas na infra-estrutura, de forma a evitar ou identificar utilizações ilícitas dos recursos disponibilizados (e.g download de conteúdos ilegais), eventualmente procedendo à responsabilização dos utilizadores envolvidos.

4 Desenho

A nossa solução visa implementar uma infra-estrutura ubíqua, que faça uso de equipamento comum (e barato) que satisfaça os requisitos apresentados anteriormente. O objectivo consiste em criar uma arquitectura que dê suporte a um cenário em que os utilizadores interagem com a infra-estrutura presente através dos seus dispositivos móveis, tirando partido das aplicações e recursos existentes, e escondendo algumas das limitações dos seus equipamentos pessoais. Esta, permite não só a existência de um conjunto bastante vasto de serviços pré-definidos, mas também dá a possibilidade ao utilizador para executar as suas próprias aplicações tirando partido dos recursos disponíveis.

A arquitectura do PIPE pode ser decomposta em duas grandes componentes. Os nós de computação propriamente ditos (*infrastructure nodes*) e os dispositivos móveis dos utilizadores, que através de uma aplicação específica podem ter acesso a todos os recursos e funcionalidades fornecidos pela infra-estrutura. Esta última componente não será obrigatória já que poderão existir aplicações que não requeiram uma interacção explícita com o utilizador (ou o seu dispositivo) ou que inclusivamente usem outras técnicas (e.g. interacção gestual) para o fazer.

Tipicamente um utilizador que disponha apenas do seu dispositivo móvel com a aplicação cliente instalada acede ao sistema através da ligação a um *infrastructure node*. Após estar ligado ao sistema, o cliente passa a poder utilizar as funcionalidade disponibilizadas (não só os serviços existentes em qualquer nó da infra-estrutura, mas também a possibilidade de instalar novas aplicações).

De seguida, detalham-se as componentes do sistema.

4.1 Infrastructure Node

O *infrastructure node* corresponde a uma máquina presente na infra-estrutura e na qual podem executar serviços e aplicações. Esta componente foi implementada usando a linguagem Java, e pode dividir-se em três módulos (fig. 1): módulo de comunicação, gestor de aplicações e gestor de serviços.

O **módulo de comunicação** é responsável por gerir as comunicações com os clientes móveis, sendo actualmente suportadas comunicações via bluetooth ou TCP/IP (sobre Wi-Fi, GPRS, etc.)

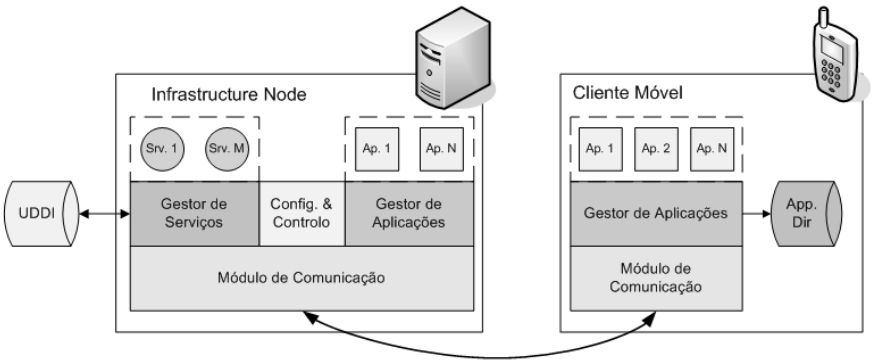


Figura 1. Arquitectura interna do infrastructure node e do cliente móvel

O **gestor de aplicações** é responsável por fazer toda a gestão das aplicações residentes na infra-estrutura e das aplicações transferidas pelos utilizadores. Estas aplicações podem fazer uso dos diversos serviços que são fornecidos pela plataforma (e.g. serviço de câmaras, serviço de reconhecimento de faces), implementando o sistema uma arquitectura orientada aos serviços.

Para permitir o controlo por parte do dispositivo móvel do utilizador, as aplicações podem definir uma interface Web que fica disponível quando a aplicação é iniciada, ou podem ter associada uma aplicação móvel (Java 2 ME) que é instalada automaticamente caso seja necessário.

Todas as aplicações executam num contexto protegido, tendo associado um conjunto de permissões diferentes, consoante serem aplicações da infra-estrutura ou instaladas dinamicamente. Para tal, usam-se os mecanismo de segurança do Java.

A instalação dinâmica de uma aplicação ocorre por pedido dum cliente móvel. No pedido, especifica-se um conjunto de fontes alternativas do código a instalar - tipicamente, o URL dum servidor e o próprio cliente móvel.

Finalmente, o **gestor de serviços** gere os serviços disponíveis às aplicações, sendo responsável pelo controlo dos serviços que correm localmente no nó, bem como pelo seu registo no directório global de serviços. Na nossa implementação, este directório é um servidor de registo UDDI [2], global à infra-estrutura em que o sistema executa.

Quando uma aplicação necessita dum serviço, esta envia um pedido ao gestor de serviços local ao *infrastructure node* onde se encontra a correr, que por seu turno verifica se tal serviço consta no directório. Se tal acontecer, então é feito a ligação entre o código cliente do serviço que faz parte da aplicação e o serviço propriamente dito. Na operação de pedido de serviço, a aplicação poderá especificar algumas preferências, em particular se o serviço requerido deverá ser local ao nó onde esta corre ou não. É ainda suportada a composição de serviços, pelo

que os serviços podem usar outros serviços de forma a fornecer às aplicações operações de mais alto nível.

4.2 Cliente Móvel

A aplicação do sistema que executa no cliente móvel gere as aplicações dos utilizadores e as suas comunicações com a infra-estrutura. No nosso protótipo esta foi implementada em Java 2 ME. Internamente é composta por dois módulos (fig. 1): um módulo de comunicações e um módulo de gestão de aplicações.

O módulo de comunicações é responsável por gerir as ligações do dispositivo móvel com a infra-estrutura e todas as subsequentes comunicações, usando os meios de comunicação disponíveis para o efeito.

O gestor de aplicações tem duas funções principais. Primeiro, é responsável por manter as aplicações dos utilizadores. Quando iniciadas pelo utilizador, estas aplicações podem executar localmente ou na infra-estrutura. Caso seja necessário executar as aplicações (ou parte delas) na infra-estrutura, este módulo é responsável por interagir com a infra-estrutura, de forma a iniciar a aplicação num *infrastructure node* apropriado.

Segundo, é responsável por contactar a infra-estrutura e apresentar ao utilizador a lista de aplicações que estão disponíveis. Como se explicou anteriormente, o acesso a uma destas aplicações pode-se efectuar usando uma interface Web ou através da execução duma aplicação móvel instalada automaticamente.

5 Aplicações de teste

Nesta secção apresentam-se duas aplicações demonstrativas da utilização do sistema desenvolvido.

5.1 Aplicação de publicidade

A primeira aplicação tem como objectivo apresentar publicidade numa zona comercial (e.g. centro comercial). Para tal, assume-se que existe um conjunto de *infrastructure nodes* distribuídos pelo espaço. Adicionalmente, existem alguns ecrãs que podem ser usados para apresentar informação e que são controlados pelo sistema. Finalmente, existem câmaras de vigilância que cobrem a zona comercial.

Para que um sistema deste tipo seja o mais eficaz possível, a aplicação de visualização de publicidade deve usar a informação sobre os utilizadores detectados nas proximidades dos diferentes nós da infra-estrutura para adequar os conteúdos mostrados ao público presente [9]. Neste caso, a escolha dos anúncios tem como base as características do local onde o nó se encontra e o histórico da deslocação dos utilizadores no espaço.

A publicidade pode ser mostrada de duas formas distintas, num ecrã público associado ao *infrastructure node*, ou directamente no telemóvel do utilizador. Pensamos que estas duas técnicas se podem complementar na medida em que

numa situação em que são detectados poucos utilizadores (e.g 2 ou 3), pode não ser desejável mostrar a publicidade no ecrã público, já que é fácil perceber a quem é que esse anúncio se dirige. Nesta situação, consideramos que é desejável que os conteúdos sejam enviados directamente para o dispositivo móvel do utilizador. Actualmente os conteúdos mostrados assumem a forma de imagens estáticas, mas a aplicação está preparada para utilizar outros formatos multimédia.

A aplicação executa na infra-estrutura e faz uso de dois serviços: um serviço de publicidade e um serviço de adaptação de conteúdos.

O primeiro, combina um serviço de câmaras, que recolhe imagens das proximidades do nó, com um serviço de detecção de faces baseado na solução desenvolvida por Grangeiro et. al. [4], a executar num nó da infra-estrutura, de forma a contabilizar o número de pessoas num dado local. Adicionalmente é usado um serviço de detecção bluetooth que permite descobrir os dispositivos próximos do *infrastructure node*. O serviço de publicidade oferece à aplicação uma listagem dos utilizadores detectados (identificados por endereço bluetooth) e respectivas preferências com base no histórico das suas deslocações.

Esta informação é usada no contexto da aplicação para efectuar a escolha do anúncio a mostrar no ecrã público, ou caso sejam detectados poucos utilizadores, para escolher para cada utilizador o anúncio a enviar. Nesta última situação é utilizado o serviço de adaptação de conteúdos, que permite adaptar os anúncios a enviar antes de iniciar a transferência para um dispositivo móvel. No protótipo actual, uma vez que se tratam de imagens, essa adaptação consiste na redução do seu tamanho original.

Esta aplicação é exemplificativa duma aplicação que corre na infra-estrutura, sem necessidade duma interacção explícita por parte do utilizador. Adicionalmente, demonstra a criação duma aplicação que utiliza um conjunto variado de serviços, alguns dos quais executam necessariamente numa localização (e.g. detecção de bluetooth, câmara), enquanto outros podem executar em qualquer localização do sistema (e.g. serviço de adaptação).

5.2 Aplicação de informação estendida sobre produtos

Esta aplicação visa demonstrar a viabilidade de aplicações que corram as componentes computacionalmente mais exigentes na infra-estrutura. Neste caso concreto, a aplicação móvel é capaz de recolher, através da câmara do dispositivo, a informação do código de barras de um produto e enviar esses dados para a componente que reside na infra-estrutura. Esta contacta diversas fontes de dados presentes na Web e oferece um mashup dessa informação ao cliente. Após estes dados serem recebidos na aplicação móvel, o utilizador pode ver informação adicional sobre o produto em causa, bem como produtos relacionados ou vídeos que sejam relevantes.

A aplicação móvel foi construída em Java 2 ME, e pode subdividir-se em três grandes módulos: captura, pesquisa e mashup.

O módulo de captura, faz uso da biblioteca ZXing[1] para efectuar o reconhecimento dos códigos de barras. O módulo de pesquisa, recebe a informação do módulo anterior e com base nas preferências do utilizador contacta o módulo de mashup que pode correr localmente ou na infra-estrutura e é responsável pela obtenção dos dados relativos ao produto e vídeos relacionados.

No módulo de mashup, quando é recebido um pedido relativo a um produto, este contacta os serviços da Amazon [2] para obter informação relativa ao produto em causa, incluindo os produtos relacionados, e os serviços do YouTube [3] para obter vídeos relacionados.

Adicionalmente, é possível ao utilizador, efectuar o download dos diferentes vídeos fornecidos. Caso escolha fazer o download do vídeo quando o módulo de mashup corre na infra-estrutura, este faz a adaptação dos conteúdos para um formato adequado ao dispositivo móvel em causa, usando um serviço de adaptação. Mais concretamente, como se tratam de vídeos o serviço de adaptação usa a biblioteca ffmpeg[4] para o efeito.

6 Avaliação

Nesta secção avalia-se o benefício de executar parte duma aplicação na infra-estrutura, usando como teste a aplicação de informação estendida sobre produtos. Os resultados apresentados comparam o acesso à mesma informação no dispositivo móvel, acedendo directamente às fontes de dados ou transferindo parte da aplicação para a infra-estrutura.

Foram realizados dois testes distintos que mediram o tempo até à apresentação da informação e a quantidade de dados transferidos no processo. No primeiro teste, a informação acedida era relativa a um produto. No segundo teste, a informação acedida consistia num vídeo.

6.1 Contexto da experiência:

Nos testes foram utilizados os seguintes equipamentos. Como cliente móvel usou-se um telemóvel Nokia N82, com o sistema operativo Symbian OS 9.2. Neste caso, usou-se apenas um *infrastructure node*, a executar num PC equipado com um processador AMD Turion 64x2, 2GB de RAM e Windows Vista. O computador estava equipado com um adaptador bluetooth (versão 2.0 EDR), um adaptador wi-fi (802.11b/g) e com uma ligação à Internet de 20Mbps, através de wi-fi.

6.2 Testes

Teste 1: O objectivo do primeiro teste foi avaliar o tempo que demora a mostrar a informação relativa a um produto, quantificando os dados que foram transferidos para o dispositivo móvel e para a infra-estrutura (caso se aplique). Esta

[1] http://code.google.com/p/zxing/
[2] http://docs.amazonwebservices.com/AWSEcommerceService/2005-03-23/
[3] http://code.google.com/intl/pt-PT/apis/youtube/overview.html
[4] http://www.ffmpeg.org

	Tempo (ms)	Transferências (KB)	
		Infra-estrutura	Móvel
Stand-alone	28441	0	147546
Bluetooth	2442	147546	13256
Wi-fi	4671	147546	13256

Tabela 1. Tempos médios de execução e quantidade de dados transferida no decorrer do primeiro teste.

informação consiste numa imagem do produto, o seu nome e respectiva classificação, uma listagem de outros produtos relacionados, bem como informação detalhada sobre o vídeo sugerido (título, duração e imagem) para além de uma lista de outros videos relacionados. Este teste foi efectuado em três modalidades distintas, primeiro com todo o processamento efectuado no dispositivo móvel. Seguidamente usando a infra-estrutura para contactar os diversos serviços e transferindo os resultados através de bluetooth e por último, usando também a infra-estrutura mas transferindo os dados via wi-fi. Os resultados obtidos apresentam-se na tabela 1.

Estes resultados mostram uma redução considerável do tempo necessário a obter a informação pretendida quando se usa a infra-estrutura. Esta redução no tempo de resposta fica a dever-se a dois factores: por um lado a transferência de uma menor quantidade de dados para o dispositivo móvel, reduzindo por isso o peso das comunicações. Este facto deve-se a que apenas uma parte da informação obtida nos serviços web acedidos é relevante para a aplicação. Por outro lado, o processamento desses dados, que são XML, requer algum poder computacional e por isso também é mais moroso no cliente móvel, sendo bastante mais rápido quando efectuado na infra-estrutura.

Teste 2: O segundo teste avalia os potenciais ganhos de se usar a infra-estrutura para efectuar a adaptação de conteúdos com alta qualidade, antes destes serem transferidos para o dispositivo móvel. Este teste obrigou a uma ligeira alteração da aplicação, na medida em que considerámos mais interessante efectuar as medições com base no download de um vídeo em alta definição relativamente a vídeos do YouTube. À semelhança do teste anterior efectuámos esta experiência em três modalidades distintas. Na primeira descarregámos directamente o vídeo para o dispositivo móvel, ao passo que nas seguintes, o dispositivo móvel contactou a infra-estrutura que por seu turno efectuou o download e adaptou o vídeo ao dispositivo móvel. Na segunda experiência a transferência dos conteúdos adaptados foi feita usando bluetooth, ao passo que na terceira usámos Wi-Fi. Na tabela 2 encontra-se o resumo dos dados obtidos.

Mais uma vez, estes resultados mostram uma redução considerável do tempo necessário a obter a informação pretendida quando se usa a infra-estrutura. Os resultados obtidos confirmam que as comunicações para o telemóvel incorrem num custo considerável, mesmo quando se usa Wi-Fi. Como se pode verificar, para o download do mesmo ficheiro de cerca de 108MB, o tempo de transferência

	Tempo (s)				Transferências (MB)		
	Total	Download	Transcoding	Transf.p/ móvel	Total	Infra-estrutura	Móvel
Stand-alone	**351,81**	351,81	0,00	0,00	**107,91**	0,00	107,91
Bluetooth	**222,56**	54,88	98,21	69,47	**112,88**	107,91	4,97
Wi-fi	**178,58**	54,88	98,21	25,49	**112,88**	107,91	4,97

Tabela 2. Tempos médios e quantidade de dados transferida para as várias entidades no segundo teste.

na infra-estrutura é cerca de um quinto. Este teste evidencia também o potencial da utilização de um serviço de adaptação de conteúdos, já que neste caso concreto, foi possível transformar um vídeo com alta qualidade e com cerca de 108MB, num vídeo em formato MPEG-4 com cerca de 5MB, sendo obviamente a sua transferência para o dispositivo móvel muito mais rápida, mesmo quando se considera o tempo necessário à adaptação dos dados, a qual na nossa solução apenas pode ser iniciado após concluir o download completo do ficheiro para a infra-estrutura.

Em suma, os resultados obtidos confirmam que a utilização da infra-estrutura para executar as componentes da aplicação mais exigentes em termos computacionais, ou de comunicação traz efectivamente vantagens. Essas vantagens traduzem-se numa redução bastante grande do tempo de resposta das aplicações e a redução das comunicações efectuadas pelos dispositivos móveis, o que se traduz igualmente numa potencial redução da energia consumida por estes dispositivos.

7 Conclusões

Neste documento descreveu-se o PIPE, uma infra-estrutura genérica de serviços para ambientes de computação ubíqua. O PIPE permite que utilizadores equipados apenas com dispositivos móveis acedam aos recursos e aplicações disponibilizadas por uma infra-estrutura de computação ubíqua disponibilizada num espaço público. Ao contrário da aproximação normal neste tipo de infra-estrutura, o PIPE permite ainda que um utilizador execute as suas próprias aplicações, ou parte delas, na infra-estrutura.

As aplicações de teste apresentadas mostram que o PIPE poderia ser usado num contexto de espaço público (e.g. centro comercial). Os resultados quantitativos obtidos mostram o benefício de executar parte das aplicações dos utilizadores na infra-estrutura, permitindo beneficiar da vantagem de uma maior capacidade de computação sem restringir o utilizador ao conjunto de serviços que a infra-estrutura disponibiliza.

Dado que o sistema PIPE se encontra ainda em desenvolvimento, como é natural, existem ainda algumas áreas a contemplar de forma a tornar esta solução mais completa. Em particular, a questão da segurança e autenticação dos utilizadores foi abordada apenas de forma preliminar. Outros aspectos interessantes

a abordar prendem-se com a utilização de diferentes técnicas para efectuar o emparelhamento do cliente móvel com o *infrastructure node* (e.g. utilizando a câmara do dispositivo móvel), ou com a partilha de ecrãs públicos entre vários utilizadores.

Referências

1. International telecomunications union - statistics, 12 Jun. 2009. `http://www.itu.int/ITU-D/ict/statistics/`.
2. OASIS - Committees - OASIS UDDI Specifications TC, 9 Jun. 2009. `http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm`.
3. Sachin Goyal and John Carter. A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *WMCSA '04: Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications*, pages 186–195, Washington, DC, USA, 2004. IEEE Computer Society.
4. Filipe Grangeiro, Rui Jesus, and N. Correia. Face recognition and gender classification in personal memories. In *ICASSP 2009 - 34th IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 04 2009.
5. M. Karam, T. Payne, and E. David. Evaluating bluscreen: Usability for intelligent pervasive displays. *Pervasive Computing and Applications, 2007. ICPCA 2007. 2nd International Conference on*, pages 18–23, July 2007.
6. Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic. People, places, things: web presence for the real world. *Mob. Netw. Appl.*, 7(5):365–376, 2002.
7. MC Lee, HK Jang, YS Paik, SE Jin, and S Lee. Ubiquitous device collaboration infrastructure: Celadon. In *SEUS-WCCIA '06: Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, pages 141–146, Washington, DC, USA, 2006. IEEE Computer Society.
8. C. Narayanaswami, D. Coffman, M. C. Lee, Y. S. Moon, J. H. Han, H. K. Jang, S. McFaddin, Y. S. Paik, J. H. Kim, J. K Lee, J. W. Park, and D. Soroker. Pervasive symbiotic advertising. In *HotMobile '08: Proceedings of the 9th workshop on Mobile computing systems and applications*, pages 80–85. ACM, 2008.
9. Terry Payne, Ester David, Nicholas R. Jennings, and Matthew Sharifi. Auction mechanisms for efficient advertisement selection on public displays. In *Proceedings of European Conference on Artificial Intelligence*, pages 285–289, 2006.
10. Jan S. Rellermeyer, Oriana Riva, and Gustavo Alonso. Alfredo: an architecture for flexible interaction with electronic devices. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 22–41, New York, NY, USA, 2008. Springer-Verlag New York, Inc.
11. M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 8(4):10–17, 2001.
12. Ya-Yunn Su and Jason Flinn. Slingshot: deploying stateful services in wireless hotspots. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 79–92, New York, NY, USA, 2005. ACM.
13. Ville Tuulos, Jürgen Scheible, and Heli Nyholm. Combining web, mobile phones and public displays in large-scale: Manhattan story mashup. *Pervasive Computing*, pages 37–54, 2007.
14. Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, 1999.

Unified Cooperative Location System

David Navalho, 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, several techniques based on location have been developed, allowing for services based on location with different degrees of precision. The possibility of using different techniques varies between devices and regions in space, according to the available hardware on the equipment and the infra-structure. This article presents the Unified Cooperative Location System, which allows the use of every available technology to a mobile device, providing a higher availability of location services. Additionally, the system implements an information exchange mechanism, allowing devices to gather location information from nearby users, like GPS or Wi-Fi, which would otherwise be unavailable for some. The possibility of exchanging GSM information provides a viable solution for the gathering of multiple GSM signals by one device, thus significantly increasing location accuracy with this technology.

Abstract. Nos últimos anos desenvolveram-se várias técnicas de localização que permitem fornecer serviços baseados na localização com diferente precisão. A possibilidade de utilizar as diferentes técnicas varia entre dispositivos e região do espaço, consoante o hardware disponível nos equipamentos e na infra-estrutura. Neste artigo apresenta-se o Unified Cooperative Location System, um sistema que permite utilizar todas as tecnologias dum dispositivo móvel para fornecer uma maior disponibilidade do serviço de localização. O sistema implementa ainda um mecanismo de troca de informação, que permite que dispositivos próximos partilhem entre si informação importante para o processo de localização que poderia não estar disponivel aos mesmos, tais como medidas Wi-Fi e GPS. A troca de informação GSM permite uma solução viável para a obtenção de múltiplos sinais GSM por um dispositivo, permitindo assim aumentar significativamente a precisão de localização usando esta tecnologia.

1 Introduction

Improvements in hardware and wireless technologies during the 90's have led to the development of multiple mobile devices, creating a new computer environment usually designated as mobile computing.

Recently, the widespread use of smaller, less expensive, and more capable devices, has opened the door for more complex applications, of which, location-aware applications are just an example [7]. Some location-based applications

are already becoming an integral part of out lives, GPS probably being the most widely publicized and used of these systems, allowing for user location and navigation.

Handheld computers are now capable of running different types of applications. Additionally, manufacturers are increasingly equipping these handheld devices with different types of wireless connectivity, including Bluetooth, Wi-Fi and sensors, such as GPS.

Taking advantage of the new characteristics and availability of these devices, several research and implementations have already been presented, in an attempt to gather all the available information surrounding a user, using it to locate himself or other individuals [1, 2, 3, 4, 5, 6]. However, even though several location systems already exist, most of these systems focus on providing location for a given user using a particular technology, without trying to leverage the advantage of the existence of multiple co-located users with different available technologies.

This paper presents our Unified Cooperative Location System, which tries to rely on every technology available to a mobile device to provide an as accurate and ubiquitous location service as possible. Our system also uses the concept of sharing location information between nearby users, in order to obtain not only better accuracies, but also to gather enough information for establishing the location. Our results show that this approach provides a viable alternative for location estimation to the end user using GSM.

The UCLS provides privacy, as well as a low cost solution by establishing the location in the users' mobile phones. Additionally, much like Placelab [1], we identified the need to provide a location framework that a programmer can use. This approach simplifies the creation of location-based applications and can be used to develop social applications to search for a friend, gaming environments, children locators, or even offer services based on the users current location [7].

The remainder of this paper is organized as follows. The following section will present the main goals of the Unified Cooperative Location System and how we aimed to achieve them. We then introduce our System's Architecture and Implementation, followed by a presentation of the early results obtained from our system. Finally, we discuss some of the existing related work and a brief conclusion.

2 Unified Cooperative Location System Principles

The Unified Cooperative Location System (UCLS) intends to provide a location solution for the end user, that has low-cost, maintains privacy and provides good accuracy. To this end, our solution runs on commodity mobile phones. Thus, unlike other solutions that require additional hardware [6], it requires no configuration or deployment overhead, and no additional costs. To maintain privacy, we rely on a client-based solution, where location is determined in the end-user device, instead of relying on an external server to continuously calculate

the user's location. For providing good accuracy, we rely on the use of all available technologies, as explained throughout this paper.

Our system uses fingerprinting [7] to build a database with readings, which is later used to determine users' location. This method requires a time-consuming process, that many people may not be willing to take on. To minimize the time required for building the needed databases, we used the same strategy as Placelab [1]. Data can be gathered automatically through wardriving (wireless information is gathered at regular intervals if a GPS device is available) and stored on the device. When an available connection to the Internet is available, the user can, if he wishes, share that information to a database, and obtain readings from every user using that database. This allows for a faster database construction.

Given the fact that mobile devices are not yet capable of heavy computations, our system allows for the construction of maps by another computer (it can be the user's own computer). The construction of the database map includes creating the data structures that allow for a fast search of the stored mappings. This is important, as it reduces the processing performed in the mobile phones when trying to determine the user's location.

Finally, our sharing mechanism allows users to anonymously share location information with other nearby users. This approach has two benefits. First, it allows to obtain better estimates by using more readings. Second, it allows users to access information their devices could not naturally obtain (e.g., a device with only a GSM connection can obtain GPS information via bluetooth from a nearby user).

Fingerprinting Fingerprinting is a simple method, where a user collects, for a set of locations, signal strength information of his surrounding wireless technologies, be it Wi-Fi APs, GSM Towers or even Bluetooth beacons. This allows for three types of locations. For Relative and Symbolic locations, there is a need for user input, where he states the location where the reading was made. For Absolute location, if GPS is available, one can automatically store the current coordinates together with the readings.

In order to locate himself later on, these readings are usually shared to other users, who do not possess either the knowledge or the technology (e.g.GPS) to locate themselves. With this information, the user locates himself as follows. First, he obtains a reading. Then, he searches the local information for the closest match, thus inferring the current position. This is possible thanks to the fact that wireless communications signal strengths do not vary significantly in a given location over time [2].

3 System Architecture and Implementation

The Unified Cooperative Location System (UCLS) consists of five Modules and a collection of Sniffers, as depicted in figure 1. The Sniffers are small and simple background services, whose purpose is to detect and gather the surrounding wireless signals and report that information to the Communications Module.

This Communications Module collects the information provided by these Sniffers and makes it available to the system. The Communications Module also contains a bluetooth service search primitive, which is mainly used by the Information Exchange Module to search for sharing-enabled devices.

The core of the system is represented by the Location Module, whose function is to control the mentioned modules, choose which information to store or retrieve from the Database Module, as well as access the available algorithms and configurations, in order to gather and provide location information. Finally, a Location API Module was conceived, so that the system can be used by a developer to integrate it easily on his developed location-aware application.

We detail the several modules in the remainder of this section, discussing not only the design but also implementation details of the UCLS prototype. Our current prototype was written mostly in Java 2 Micro Edition, with some notable exceptions that are discussed later.

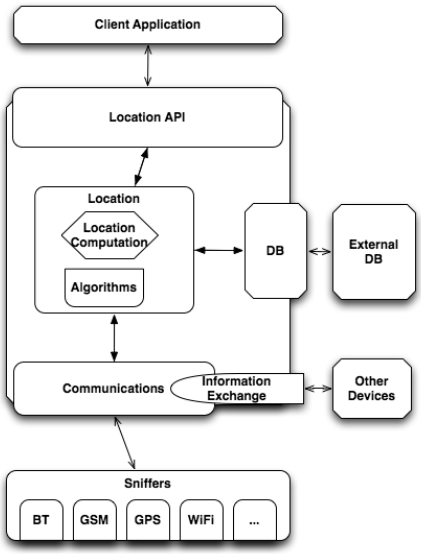


Fig. 1. Unified Cooperative Location System architecture.

3.1 Sniffers

The Sniffers basically access the Operating System's native APIs to obtain the signal strengths of the different available wireless technologies. Unlike the rest of the UCLS prototype, the sniffers were written in Python S60. This was necessary due to the limitations of the Java 2 ME system, which does not provide access to the necessary information about wireless signal strengths. As we want our system

to be extensible and modular, we have implemented an independent Sniffer for each technology. This allows to use only the sniffers for which there is hardware.

Sniffers Communication Protocol For making Sniffers information available to the rest of UCLS, Sniffers must act as a simple TCP server, and execute a basic standard communication protocol. The protocol works as follows.

Each Sniffer needs to receive an activation message to start collecting information. This is necessary because, unlike GSM information which is always available to the device at no extra cost, some technologies, such as GPS, need to be active only when location information is required. A deactivation message is also used, since some sniffers (e.g.:GPS) waste high amounts of energy while working. This allows to deactivate them when not needed, conserving battery. Other than that, the server only receives reading requests and replies back with the gathered information.

The code itself is very simple to implement, and it should be fairly quick to implement a new module, be it a new technology, or an implementation for another Operating System. In the current prototype, the available Sniffers are implemented in PyS60 (Python for Symbian S60), since it provided the necessary tools to gather the required information, while providing a very simple programming environment.

3.2 Communications Module

The Communications Module is responsible for contacting the available Sniffers and grouping the acquired information. This Module controls the communications to the Sniffers, and reports the gathered readings as well as the available technologies for location to the system.

Additionally, a set of bluetooth primitives are also available when the package for bluetooth (JSR82) is available. These primitives are mainly used to search for devices running specified services. This is used primarily by the Information Exchange Module, which uses this functionality to search for sharing-enabled devices. This functionality can also be used by the Database Module to communicate with external services or servers to obtain or store data when a Wi-Fi connection is unavailable.

3.3 Information Exchange

The Information Exchange module is responsible for exchanging location information with nearby devices. This module exports a bluetooth service that allows other devices to obtain information gathered locally. We have decided to use only bluetooth communication because of its ubiquity and short-range. This guarantees that a device can only access devices that are close enough to provide useful information.

To obtain information from nearby devices, the Information Exchange Module continually searches for available devices by using the Communications Module service search functionality. When a device available for sharing is detected,

they can exchange any kind of location information. For example, a device with less capabilities can obtain the GPS location information from a nearby device with such functionality. Additionally, a device can obtain the raw signal strength readings from the nearby devices, and use them in its local location computation. Depending on the technology and distance between the devices, this information can be used to average the error of individual readings. Additionally, by using every available reading from all devices it might also be possible to improve location accuracy. This is especially evident when devices share GSM signal information, where the presence of only two devices connected to two different GSM Towers immediately offers a more accurate result.

There are some points to take into account when implementing an information exchange service for location. The first is how useful the information can be to improve accuracy. Bluetooth can reach only a few meters at most of range, thus users must be very close to each other when sharing location information. Although this adds some error, one can still obtain improvements when the error of the location technique used is much higher than a couple of meters, like when using GSM. Unfortunately, the Symbian implementation for bluetooth does not provide the signal strength information of a bluetooth communication, which could eventually help to estimate the relative distances between users. This information could be used to lower the estimation error while sharing location information.

When sharing information, an important aspect is whether the users are moving or not. If users are moving, it is important to have the readings at the same time (or as close as possible). To this end, the service provides a reading timed request, which basically receives a time when the reading should be performed. In the current prototype, we assumed that the clock in the devices were sufficiently synchronized through the mobile operator, but some internal clock synchronization protocol could be implemented. If users are mainly stationary, the reading can be obtained immediately. To this end, the service also provides a primitive that immediately returns the value read.

3.4 Database Module

The Database Module allows other modules to store and retrieve readings information. This module stores readings and mapping information on the mobile phone by default, but it can be configured to communicate with an external server to gather or deploy new information.

Additionally, a simple mechanism was devised, to share the whole readings collection with an external server. This server can then aggregate readings from multiple users, and maps can be created based on all available information. Maps are basically data structures that are designed to make efficient the execution of the location algorithms in the mobile phones. This data structures can consist in simple listings of all readings or more complex data structures that allow for faster and more efficient searches.

Database Creation UCLS supports two approaches to add new readings to the system database: automatic and manual. The manual mapping is useful for symbolic locations, allowing to map areas such as buildings and, if the technologies are available and allow for such precision, floors, rooms or even smaller sections of rooms. The automatic location is used for wardriving-like fingerprinting, storing location information when a new GPS lock is acquired every few seconds.

Whenever the Location Module uses the Communication Module to gather location information, it then sends that information to this Database Module. Since some devices may be more limited than others, the Database Module can be set to either store the information locally or externally. Additionally, the Database has mechanisms to communicate with a server via Wi-Fi (or bluetooth if available) and upload the users' readings.

Information uploading provides two advantages: the first is the creation of an external shared database, that stores location information from several users. The second is the possibility of creating offline maps, more complex but allowing for faster searches when working with the mobile phone offline. The system currently has a very simple algorithm, where a device can communicate with an external server and download a map containing all the location information. This can be a complete map, or just a portion of it.

Map building The current implemented method to build a map for the mobile device is very simple, yet efficient. When the user requests a map from the server, the server gathers all the relevant location information stored and builds a map for the user.

The maps are nothing more than a collection of data structures that allow a quick search of a small grouping of possible current locations given a set of readings. A set of hash tables are created, each storing signal strength information ranging from a group of signal strength values. This information is accessed by the algorithm that determines user's location.

3.5 Location Module

At the core of the system, the Location Module is in charge of controlling each Module of the system, allowing for automatic location sensing capabilities, or more precise and controlled options. This module acts as the coordinator of the System, obtaining location information from the Sniffers, and contacting nearby devices, if available, to increase location accuracy. Furthermore, it uses the DB's information to read whatever maps are available, be it local or external to the device, and infer the current position using the available algorithms.

In our current prototype, we have only implemented a single location approach, based on fingerprinting. In this case, for inferring the user's location, the algorithm, based on the solution proposed in RADAR [2], gathers all the available wireless information at a given time, and it tries to find similar matches using the Database Module. Based on this information, the algorithm just guesses to be in the location of the closest match.

In the future, we expect to extend the location module to include additional algorithms, and allowing for a choice between different algorithms, based on the available information, energy remaining, etc.

3.6 Location API Module

The API Module serves the simple purpose of delivering an API for a developer to use a location system on his application. This API provides calls to enable or disable any available technology on the device (GPS, GSM, Wi-Fi), as well as a call to enable or disable the information sharing mechanism. The API also supplies the programmer with two distinct reading gathering mechanisms: manual and automatic. The manual gathering call is used mainly for symbolic gathering, although if no symbolic location information is supplied, as long as a GPS lock is acquired, it will still store the reading. The automatic is used only for absolute positioning, and can be enabled or disabled through this API.

The Location API Module also provides primitives to share the currently stored readings with an external server, as well as a primitive to request an updated map from the same server. This map can be a simple collection of signal readings, or a more complex map, as mentioned previously. Finally, the API supplies a single call for estimating the device's current location, which will automatically gather the surrounding wireless information, including nearby devices information (if that option is enabled), and infer the position with the available algorithms present in the Location Module.

4 Experiments

Although the evaluation of our system is still underway, we have already obtained interesting results regarding GSM localization, when sharing information among different users connected to different operators. We have gathered information over the course of two days for four different buildings in our campus. Three of those buildings are close together, while the fourth is no more than 100 meters distant from the other three. In order to obtain the readings, we used between three and six devices (N82 and E70 Nokia mobile phones) where we had installed our system. Each phone was connected to one of three providers, and each two connected to the same provider, were using a different type of connection (GSM and UMTS). This is important as we have observed that different connection types supply different cell ids and signal strengths.

For building the database, all the readings were collected in a single mobile phone by using our information exchange technique to obtain the reading from all devices. We then stored all the information on a database in a server. We managed to obtain a total of 450 shared readings, where each reading contained at least three identified cells. In total, there were 1958 readings from 21 distinct cell ids. The readings in some buildings include unique cell ids that make determining the location very simple. Thus, we concentrate our study in the subset

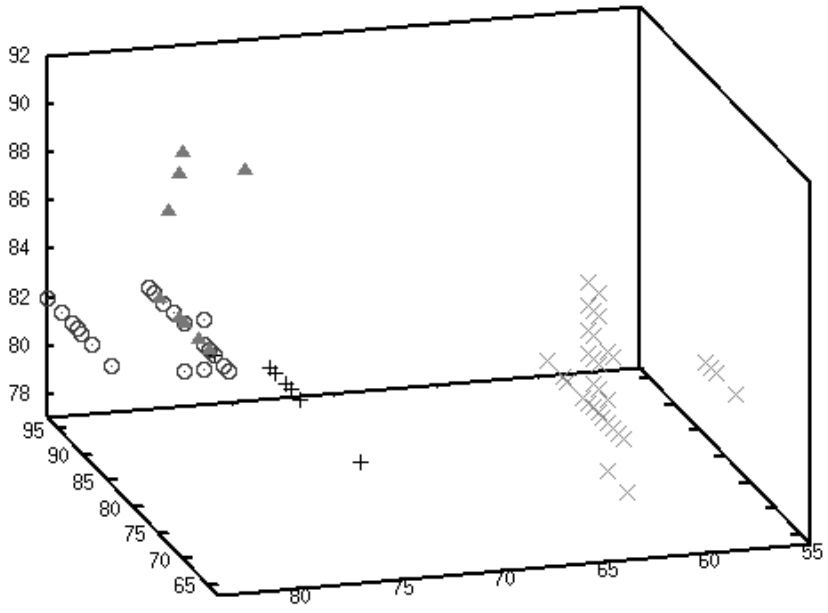


Fig. 2. 3D visualization of the three cells signal strengths. Legend: black plus signs building I, magenta crosses building II, blue circles building VII and the red triangles building X

of readings that include the same three cell ids. These readings include locations in all buildings. Figure 2 shows a graphical representation of the readings, with each axis representing the signal strength for each of the three cells. We represented the reading in each building with the same symbol and color.

A quick look at the figure allows us to clearly distinguish 2 buildings, the first relative to building II, and the second one relative to building I. The readings for the other two buildings (VII and X) are not so distant, as those buildings are located very close to each other.

After these initial observations, we used our algorithm to test the location accuracy offline. We executed the following process 10.000 times. From the complete database, we have randomly removed one reading. We applied the location algorithm to the removed reading, considering only a subset of the cell-ids obtained for the reading (e.g., if a reading includes the signal strengths for six cell-ids, we have randomly considered only n of those values). Then, we compared the estimated and actual location (considering the symbolic location consisting in the buildings). Table 1 summarizes the results obtained, representing the amount of cells used to calculate the symbolic location on the first column, and the percentage of times a location was correctly identified out of 10.000 at-

Table 1. Symbolic location accuracy using GSM

Number of Cells	Hit Rate
1	73.0
2	94.0
3	97.3
4	98.5
5	99.9
6	100.0

tempts at location. These results show that by considering a second cell, similar to sharing the information with just a single other mobile phone connected to a different cell, the accuracy improves from 73% to 93%. These results suggest that the use of our Cooperative System may allow important improvements in location estimation, even when using just GSM/UMTS technologies.

5 Related Work

There has been a lot of research on the area of location systems, and many different approaches to it. Some use specialized hardware to provide a solution [3, 6] while others use the available infrastructure [2, 4, 1] to provide users with a cost-free solution for location estimation. Amongst the available solutions, we find three main types of location: symbolic[4], relative[2] and absolute[1]. Besides using fingerprinting methods [2, 4] to locate a user, there has also been some research to use signal propagation modeling for both symbolic [5] and absolute [1] location estimation.

The cricket system [6] provides an automatic indoor mapping service, where a user needs only add a new device and set its location name, allowing for an easy to deploy symbolic location environment, though it also needs specialized equipment.

For automatic data gathering, Placelab's solution [1] introduced the concept of using wardriving to obtain location information, which also meant public wardriving databases could be used to gather maps quickly and provide absolute location. Additionally, some research has been done regarding self-mapping algorithms[9], though the map building phases have revealed to be very intensive and require an available computer with stronger computational power.

Calibree [8] uses relative location estimation to help determine users positions by sharing GPS coordinates.

GSM technology location-based solutions already have been subject of several studies [10, 11]. It has been shown that GSM can be used for symbolic location differentiating between floors and within-floor accuracy ranging from 2.5 to 5.5 meters [10] and absolute ranging from 100 to 200 meters [11]. However, this work is based upon special devices (either modified phones or special modems), which do not present a low-cost solution for GSM location. Our solution of sharing information provides a practical approach to use similar ideas.

6 Conclusions and Future Work

In this paper, we have presented a system that provides location based on any wireless technology available to a mobile device, effectively providing a unified solution that can take advantage of several algorithms. Unlike previous solutions, our approach allows a device to share its location information, both raw and processed, with nearby devices. This may allow a device to improve location results, both by using information from technologies that are not available in the device and by using additional readings. For example, this approach makes the Unified Cooperative Location System a practical solution for GSM based location, by recruiting the help of other nearby devices and obtaining multiple cell readings, thus improving accuracy compared to systems that can only obtain a single GSM reading, as is usual on most mobile phones.

We presented early evaluation results of our system, starting with the technology that takes the most advantage of these shared readings, GSM. We are planning to further extend our research of Wi-Fi and GSM location as well as attempt to determine how to best take advantage of simultaneous technologies for location inference.

In the future, we also plan to add Relative location estimation to our system, as well as try to use mobile devices or computers that can calculate a Bluetooth's connection signal strength, to determine relative distances between devices. Additionally, it would be interesting to add signal propagation modeling [12] to the system, and try to measure our distance to an AP or GSM Tower based on the signal strength.

If mobile operators would provide, besides cell information, the absolute coordinates of the tower the device is connected to, a user could use our information-exchange mechanism to share tower information between nearby users and triangulate his position based on the signal strengths of each tower.

References

[1] Lamarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tabert, J., Powledge, P., Borriello, G., Schilit, B.: Place Lab: Device Positioning Using Radio Beacons in the Wild. (2005)

[2] Bahl, P., Padmanabhan, V.N.: Radar: an in-building rf-based user location and tracking system. In: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Volume 2. (2000) 775–784 vol.2

[3] Harter, A., Hopper, A., Steggles, P., Ward, A., Webster, P.: The anatomy of a context-aware application. Wirel. Netw. **8**(2/3) (2002) 59–68

[4] Varshavsky, A., Lamarca, A., Hightower, J., de Lara, E.: The skyloc floor localization system. In: Pervasive Computing and Communications, 2007. PerCom '07. Fifth Annual IEEE International Conference on. (2007) 125–134

[5] Ji, Y., Biaz, S., Pandey, S., Agrawal, P.: Ariadne: a dynamic indoor signal map construction and localization system. In: MobiSys 2006: Proceedings of the 4th

international conference on Mobile systems, applications and services, New York, NY, USA, ACM Press (2006) 151–164
[6] Priyantha, N.B., Chakraborty, A., Balakrishnan, H.: The cricket location-support system. In: MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking, New York, NY, USA, ACM Press (2000) 32–43
[7] LaMarca, A., de Lara, E.: Location Systems: An Introduction to the Technology Behind Location Awareness. Morgan & Claypool Publishers (2008)
[8] Varshavsky, A., Pankratov, D., Krumm, J., Eyal de Lara: Calibree: Calibration-free localization using relative distance estimations. In: Proceedings of the 6th International Conference on Pervasive Computing (Pervasive). Volume 5013 of LNCS., Springer (2008) 146–161
[9] Lamarca, A., Hightower, J., Smith, I., Consolvo, S.: Self-mapping in 802.11 location systems. In: Proceedings of the Seventh International Conference on Ubiquitous Computing (Ubicomp 2005), Lecture Notes in Computer Science, Springer-Verlag (2005) 87–104
[10] Varshavsky, A., de Lara, E., Hightower, J., LaMarca, A., Otsason, V.: Gsm indoor localization. Pervasive Mob. Comput. **3**(6) (2007) 698–720
[11] Chen, M., Sohn, T., Chmelev, D., Haehnel, D., Hightower, J., Hughes, J., Lamarca, A., Potter, F., Smith, I., Varshavsky, A.: Practical metropolitan-scale positioning for gsm phones. In: UbiComp 2006: Ubiquitous Computing. (2006) 225–242
[12] Valenzuela, R.: Ray tracing prediction of indoor radio propagation. In: Wireless Networks - Catching the Mobile Future., 5th IEEE International Symposium on. Volume 1. (Sep 1994) 140–144

Towards a Context Aware Multimodal Hand-Held Device

Tiago Reis, Carlos Duarte, Luís Carriço, Romeu Carvalho

LaSIGE, Faculdade de Ciências, Universidade de Lisboa

{treis, romeu.carvalho}@lasige.di.fc.ul.pt, {cad, lmc}@di.fc.ul.pt

Abstract. We present the design of a prototype for a context aware multimodal hand-held device, envisioning several improvements on the interaction with such devices. The design of this prototype is strongly based on an extensive requirements and feasibility analysis. Different context awareness and multimodal interaction issues are depicted, based on previous and actual user centered experiments. We concluded that the sets of sensing mechanisms available on commercialized devices are not enough for our aspirations and that the augmentation of such devices through the utilization of sensor platforms is not applicable on real life scenarios, leading us to design a capable device for such purposes.

Keywords: Requirement and Feasibility Analysis, Hand-held Devices, Context Awareness, Multimodal Interaction.

1 Introduction

Researchers have been focused on context since the 90's. Many definitions emerged for this concept [1]. However, one is abstract enough to include all the others and to keep a spectrum wide enough to cope with technology's evolution: "Context is any information that can be used to characterize the situation of an entity" [2]. An entity may be a person, an object, or anything that is characterizable. Context awareness defines the capability of applications to access information regarding their utilization context and react accordingly. Underneath it, locally or remotely, context sensing defines the capability of a device or infrastructure to sense context through the utilization of sensing mechanisms (e.g. sensors, cameras, microphones, etc.). Once available, context information can be exploited for different purposes [2].

Besides enabling context awareness, sensing mechanisms may also be employed in order to support input modalities (e.g. voice recognition, acceleration-based gestures, computer vision-based gestures, etc.), augmenting multimodal interaction. Multimodal interaction is a characteristic of everyday human activities and communications, in which we speak, listen, look, make gestures, write, draw, touch and point, alternatively or at the same time in order to achieve an objective. Achieving a more natural human-computer interaction, by considering the human channels of perception and communication through the inclusion of elements of natural human behavior on human-computer interfaces, is the main goal of multimodal interaction [3].

Context awareness and multimodal interaction are two concepts that are strongly related. The ubiquitous nature of mobile and hand-held applications increases the number and complexity of the contexts in which they are used. This complexity may limit or even inhibit interaction [4, 5]. However, multimodal interfaces can improve interaction for different users and usage contexts, increasing performance, stability, robustness, expressive power, and efficiency of mobile activities [6, 7]. Therefore, the interaction limitations introduced by context may be overcome by multimodality. The individual or assembled utilization of different interaction modalities, which may be automatically or semi-automatically enabled, disabled, configured, and optimized according to the different variables that define a usage context, may introduce significant improvements on human interaction with hand-held devices.

To achieve the constant and reliable context awareness desirable for this purpose, the use of remote context sensing must be minimized. The principle itself seems inadequate, and therefore very hard to conceive, in macro scale solutions, such as context-aware cities, countries or continents. Moreover, the macro scale implementation of such concept would almost certainly raise a social dilemma regarding the users' privacy. One plausible solution is the inclusion of sensing mechanisms on hand-held devices. On one hand, this would allow the majority of the context sensing and interpretation routines to be performed by the device and, on the other, it would enable richer interaction experiences for their users. Nevertheless, the integration of these devices with context-sensing infrastructures and the internet should also be considered, broadening the employments of both context information and interaction modalities.

This paper focuses on the relations between sensing mechanisms, context-awareness, and multimodality aiming several improvements on human interaction with hand-held devices. The next section introduces the background and motivation for the work presented. Following, we describe the requirements and feasibility analysis of our context-aware multimodal hand-held prototype. Some of the improvements suggested are validated referencing several experiments and others through our own experiments, which are further described. A sketch of the prototype is presented and, finally, conclusions and future work directions are articulated for this on-going research.

2 Background and Motivation

The first contribution for the mobile context awareness domain was the Active Badge system, which integrated a positioning technique with a distributed computing infrastructure [8, 9]. Following, the Ubiquitous Computing vision pointed out the importance of location and context for the next era of computing [10]. These contributions underlie the ParcTab experiment, where hand-held devices were augmented with locality in order to access location-based services [11, 12]. Nonetheless, these pioneer projects all employed indirect awareness, with sensors located on an infrastructure responsible for the context sensing. These infrastructures have proven to be very useful in controlled environments such as museums, art galleries and tour guides [13, 14], presenting an enormous potential for many others

such as hospitals, enterprises and offices [15]. However, the fully infrastructure-based context awareness paradigm does not seem to scale in order to achieve macro scale solutions (e.g. context aware city).

Due to the recent boost in sensing technologies, researchers increased their interest in embedding direct awareness into hand-held devices [16]. The trend is leading to sensors in small packages that enable powerful sensing at very low cost. The amount of context information that can be sensed locally on these devices is considerably larger than before and in constant growth. Actual hand-held devices are significantly more powerful and integrate more sensing mechanisms. Experiments conducted show that context information can be gathered locally and constantly by hand-held devices. Location [8, 9], orientation [17], lighting, temperature [18], noise [19], posture [20], movement [21], activity [22], and grasp [23] have been accurately identified and explored for different purposes. Nevertheless, these were mostly laboratory experiments conducted in order to prove concepts. Regardless of the growing availability of sensing mechanisms on commercialized hand-held devices (e.g. cameras, accelerometers, g-sensors), the available sets have proven to be insufficient for the goals here proposed. Some researchers augmented existing devices employing sensor platforms that enable the exploitation of different sets of sensing mechanisms [24, 25, 26]. Despite the good results achieved, the approach does not seem adequate for real life scenarios. Users do not want a sensor platform attached to their devices on their day-to-day lives. Moreover, the simple fact that these platforms are attached to the devices may introduce interaction difficulties.

Available studies [5, 27] show the impact of context in mobile and hand-held interaction, suggesting the creation of intelligent applications that react to context, adapting the modalities provided for interaction, accordingly. Other studies employ these suggestions in order to improve HCI on different aspects (e.g. automatic noise-based volume adjustments) [19, 21]. Even some commercialized devices start to employ these concepts (e.g. adjusting graphical user interfaces according to screen orientation, adjusting brightness and contrast according to the environmental light).

Facing the amount of context information that is technologically possible to acquire on hand-held devices nowadays, the actual employments of the concept seem rather simple. Due to the lack of devices integrating a reliable and robust set of sensing mechanisms, most experiments conducted consider and employ different context variables separately. The potential inherent in the merging of such contextual aspects broadens the spectrum of human interaction with hand-held devices. This motivated the design of the prototype further presented in this paper. This prototype envisions a reliable, robust and technologically feasible solution for local context awareness. It aims at several improvements on human interaction with hand-held devices, through the utilization of different sensing mechanisms, and focuses 3 main topics: 1) acquisition and interpretation of context; 2) creation and utilization of sensing based input modalities; 3) contextual adaptation of interaction modes and modalities.

3 Prototype Design

This section starts with a description of the requirement and feasibility analysis conducted in order to enable the design of a context-aware multimodal hand-held device. Afterwards, the envisioned device is presented and described.

3.1 Requirement and Feasibility Analysis

In order to design the prototype proposed on this paper, requirements and feasibility proofs were gathered from the data assembled on ours' and others' experiments. The experiments available on the bibliography are just referred, while new experiments conducted are presented and discussed. The process focused essentially the relations between:

Sensing mechanisms and context variables. Sensing mechanisms enable the interpretation of context variables (e.g. a microphone enables noise-awareness). In some cases, the values of specific context variables may be employed in order to infer additional context information, especially if the temporal dimension is also considered (e.g. inferring velocity from a location historic). The accuracy of the contextual information gathered is defined by the accuracy of the sensing mechanisms used, the length of their inactivity periods, and the accuracy of the inference methods used.

- GPS is an accurate technology, which is generally used in order to enable location-awareness on outdoor scenarios. The device's location indicates its user's location; velocity may be estimated based on a location historic; and, movement can be inferred from velocity [28]. For instance, it is plausible to assume that different average velocities correspond to different movement states (e.g. stopped, walking, running, in a vehicle).
- Infrared technology has been employed successfully and accurately in order to enable location-awareness on indoor scenarios. Again, user location, velocity and movement may be inferred [15].
- Cell ID has also been employed in order to enable location-awareness. The advantage of this approach is its suitability regarding both indoor and outdoor scenarios. However, the approach is significantly less accurate than the previous two and, accordingly, the context information that may be inferred from location will also be significantly less accurate [29].
- RFID technology was successfully and accurately employed in order to enable indoor location-awareness. This technology enables entity-presence- awareness as well [18].
- Magnetic sensors (e.g. electronic compass) enable precise orientation-awareness [28].
- Microphones enable environmental sound-awareness. The sound captured may contain indicators of noise, existence of ongoing activities, and presence of other sound emitting entities (e.g. people, vehicles, machines). In this case the accuracy of the information gathered also depends on the quality and number of microphones used [19].

- Cameras enable lighting-awareness and computer vision based entity-presence-awareness [31].
- Photo sensors enable lighting-awareness [18].
- 3D accelerometers provide awareness regarding user velocity, movement [22] (e.g. stopped, walking, running), posture [20] (e.g. sitting, standing), and orientation [30]. These sensing mechanisms also supply awareness regarding the device's screen orientation [32] and position [20] (e.g. pants pocket, jacket pocket, bag).
  We envisioned the possibility of using accelerometers to infer the ear in use when one is having a mobile phone conversation. In order to prove this concept, an experience involving 4 users was conducted. The experiment considered real usage scenarios defined by: 1) different movement states (sitting, standing, walking, and running); 2) distinct initial positions of the device (hand, pants' pocket, jacket pocket, bag and belt); and, 3) on-conversation ear switching. The methodology implemented presented accuracy rates of 100% on all scenarios and the possible applications of ear-awareness are presented on the description of the relations between context variables and interaction.
- Temperature and humidity sensors enable temperature and humidity-awareness [18]. Humidity sensors were left out the prototype design due to their lack of utility on most hand-held interaction scenarios.
- Matrixes of capacitive sensors wrapped around a hand-held device enable grasp-awareness. Experiments show that this approach is able to accurately distinguish 5 different classes of grasp [23]. However, these sensors were excluded from the prototype's hardware requirements. We believe that the same principle can be implemented through the utilization of matrixes of pressure sensors, with the advantage of introducing expressiveness into the grasp-awareness process.
- Wi-fi and Bluethoth technology may be used in order to enable entity-presence-awareness within the signal range (e.g. devices of other users, available services). Moreover, the technology enables the use of existing internet connections that may be used to access contextual information available on the web (e.g. weather). In this particular case the accuracy of the contextual information gathered is strongly influenced by discrepancies between the context of the sensing entity and the context of the device employing it.
  We developed a simple application that enables users to create contextual reminders, a simplified infrastructure-independent version of [15]. The application enables users to associate reminders to devices that represent other users. When these other users are within the signal range, the application presents the previously defined reminders. Four users tested this application on their day-to-day lives during one week and all of them reported the immense utility of such application.

Sensing mechanisms and input modalities. Sensing mechanisms may also be used in order to support and/or augment input modalities.

- Microphones underlie the support of audio recording, voice recognition, and blow interaction.
  In order to prove the concept underlying the latter, blow interaction was integrated on two well-known game applications: Pong and Space Invaders. The

first application was object of two experiments, involving 5 users each. On both experiments users had to use blow in order to move the Pong Bar on the screen. The first experiment considered the use of only one microphone. The Pong Bar position was influenced by gravity and by a force with gravity's inverse direction. Blow intensity was used in order to compute this second force. All the users involved mentioned an increased fun factor when using this modality. However, all of them emphasized the fact that the experience was exhausting and made them feel dizzy. The second experiment considered the utilization of two microphones. Again, forces were computed according to the blow intensity captured on each microphone. However, this time gravity did not influence the position of the Pong Bar. All users agreed that this type of interaction was as fun as the previous, although less exhausting.

The second application, Space Invaders, was object of one experiment involving 4 users. Blow interaction was used in order to fire against enemy spaceships. All the users involved reported that, in this case, fun factor increased significantly comparing to the traditional interaction modalities (e.g. keypad, touch) and that the experience was slightly tiring.

- Cameras may be used in order to record video, take pictures, and enable computer-vision-based interaction modalities [33] (e.g. gesture recognition, face recognition, expression recognition, object recognition).
- Photo sensors may also be used as an input mechanism. The spectrum of values between complete darkness and the average environmental light at a specific moment may be mapped into an interval of values that provide input to an interface control (e.g. button, scroll bar).

  In order to prove this concept, two simple applications were implemented and experimented by 4 users. Both applications mapped lighting information to scroll input. On the first application two photo sensors were used as buttons, one for scrolling up and the other for scrolling down the document. Whenever the lighting information on one sensor was close to dark the correspondent button would be pressed. All the users involved found this approach very easy to use and applicable on real life scenarios where environmental lighting is stable.

  The second application used only one photo sensor in order to compute the location of the scroll bar. Three of the users involved found the concept easy to use and applicable on situations where they just had to get a general view of the document. However, for time consuming tasks, like analyzing and commenting a document, the approach was considered difficult to use due to its lack of state memorization (e.g. the user must be constantly covering the sensor in order for the document to stay in one place).
- 3D Accelerometers enable acceleration and tilt interaction, both on 3 different axes. The variables considered were employed with success, enabling acceleration-based gesture interaction [34] and tilt interaction [17]. Actually, there are some commercialized devices that integrate such technology and exploit it for the latter mentioned purpose.
- Pressure sensors and pressure sensitive touch screens improve touch interaction, introducing pressure as a new interaction dimension. Accordingly, touch based interaction modalities (e.g. gesture recognition) may also be augmented with pressure, increasing their expressiveness.

In order to prove this concept a simple application was developed and experimented by 4 users. Interaction was supported by two pressure sensors responsible for zooming images in and out. The first experiment ignored the pressure information, using the pressure sensors as simple touch sensors. The second experiment considered pressure as an input parameter, which was mapped into zooming velocity. These experiments shown that pressure sensors may be used as simple touch sensors, and that the added pressure dimension was very intuitive and useful on the opinion of all the users involved.

- Magnetic sensors may be used to create a style of portable augmented reality interaction [17]. The concept of augmented reality relies on the handheld's display providing a moveable window into a virtual world which is somehow correlated with the real world. This metaphor tries to provide the best augmented reality possible without a special infrastructure for sensing orientation.

Context variables and interaction. Context information has a strong impact on interaction. By itself, this information already introduces interaction improvements (e.g. presenting user location and orientation on a map). However, it may also introduce interaction difficulties (e.g. environmental noise may limit or inhibit voice communication or recognition). On a device that supports several interaction modalities, context information may be employed in order to improve interaction, automatically or semi-automatically adjusting and configuring the interaction modes and modalities available. The following list presents simple examples that materialize this concept, opening doors for examples of increased complexity.

- Location-awareness is very important on the support of egocentric map applications, such as tourist guides, where the graphical representation of the user's location plays a paramount role [28]. Moreover, location information may be employed in order to discover services and automatically access information related to the user's location [35].
- Orientation-awareness is also vital for most location-aware egocentric map applications, where the user's orientation must be graphically represented [30, 28]. Moreover, it enables hand-held devices to act as virtual pointers to services and information anchored at specific entities, such as regions, buildings, and landmarks [35].
- Entity-presence-awareness enables the graphical representation of entities on egocentric map applications and access to entity related information [28].
- Screen-orientation-awareness enables the automatic adjustment of graphical user interfaces [32]. In fact, some commercial devices already implement this concept.
- Grasp-awareness may be employed in order to adjust different aspects of applications' GUIs [23] (e.g. location, size, and sensibility of the GUI controls).
- Noise-awareness enables automatic adjustments of different aspects of audio interaction. Regarding input modalities, noise can be measured in order to predict voice recognition's accuracy rates, enabling applications to suggest users the utilization of alternative input modalities in case these accuracy rates are below a certain threshold. Concerning output modalities, noise information may be employed on automatic volume adjustments directed to different aspects of audio output (e.g. ringtone volume [36], headphones and earphone volume [19]).

- Lighting-awareness enables automatic adjustments of visual-related interaction modalities. Regarding input modalities, and considering the use of a touch screen, the value of environmental light may be employed to activate voice recognition in cases where the reflection on the screen inhibits the graphical interface visualization. Regarding output modalities, screens' brightness and contrast may be automatically adjusted in order to improve graphical output quality [37]. Some commercial hand-held devices already include the latter mentioned functionality and the sensing mechanisms that support it.
- At first sight the employment of temperature-awareness might seem useless regarding interaction improvements. However, if we consider very cold scenarios where users have to interact with virtual keyboards through touch (e.g. writing a text message), the employment of such information starts to make sense. Very cold temperatures usually imply the use of thick gloves, which introduce difficulties on touch interaction.
  In order to attest the validity of employing temperature information towards improving human interaction with hand-held devices, a small experiment was conducted. The experiment involved 4 users, which had to write two text messages of similar lengths while wearing snow gloves. The first task considered the use of a regular virtual keypad, while the second considered the use of a virtual keypad with larger keys (50% larger). The results present an average mistake reduction of 60% and an average time improvement of 40%, proving the feasibility and applicability of temperature-based GUI adjustments on touch-based interaction.
- Ear-awareness describes the ear through which a user is communicating during a phone call. This information may be used in order to compensate little hearing unbalances through automatic volume adjustments. However, our implementation of this concept was not evaluated from a medical point of view.
- Awareness regarding a user's velocity, movement, posture, and his/her device's position enables automatic configuration adjustments [21] (e.g. set a mobile phone to vibrate if the user is sitting with the device on his/her pocket, or set a mobile phone to ring loudly if the user is running with the device on his/her backpack). Moreover, according to these same contextual variables, certain applications may be initiated automatically (e.g. a health control application for joggers if the user is running) and provide more adequate interaction configurations according to different movement states (e.g. if a user is running and wants to write an SMS, the application may provide that user with the means to dictate that message instead).

Interaction, users' preferences and capabilities. Users' preferences and capabilities play paramount roles on the automation of interaction configurations and optimizations. The extreme importance of these two contextual dimensions on human-machine interaction suggests the utilization of user profiles. Such profiles should include information regarding users' preferences and capabilities, allowing users to define and/or configure contextual-based interaction rules. The automatic completion of this profile could also be achieved through machine learning techniques. Both these concepts were considered and employed successfully in ours' and others' previous research [19, 23]. However, further studies are necessary in order

to scale these concepts to the amount of context information and interaction modalities here proposed.

3.2 Prototype Sketch

The fusion of a set of sensing mechanisms, which, together, are capable of reliably enabling the inference of the previously exposed context information, as well as the support the above mentioned input modalities, will, presumably, improve human interaction with hand-held devices significantly. Figure 1 presents the sketch of a prototype envisioned for this purpose, enhancing the sensing mechanisms used on each side of the device. Besides these mechanisms, this prototype should incorporate a 3D accelerometer, a GPS, an electronic compass and an internal antenna for phone communication, Bluetooth and Wi-Fi connections.

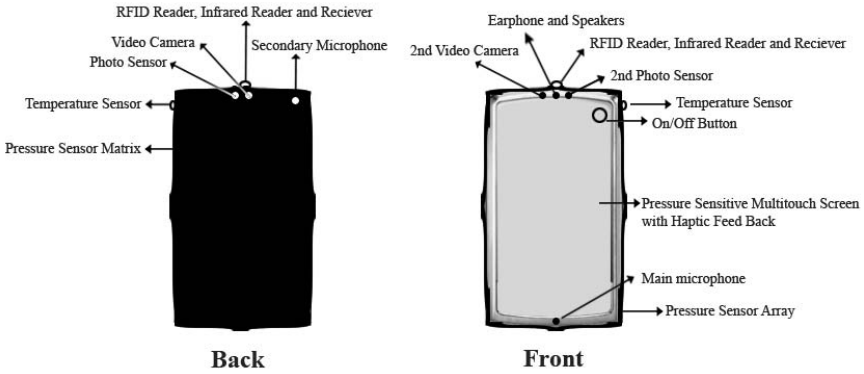


Figure 1. A context-aware multimodal hand-held device.

Most of the components available on this device were selected according to the previously presented requirement and feasibility analysis. However, some details require further justification:

Two photo sensors. The decision of including two photo sensors on our device is based on the fact that, during user interaction (e.g. answering a call) one photo sensor might be covered, presenting values that are not in accordance with the environmental light at the moment of interaction.

Two cameras. We envision the inclusion of two cameras on our device. The camera available on the front side of the device enables users to engage video conferences, while seeing the other user(s) on the screen. The camera available on the backside enables the usage of the device as a regular video/photo camera, taking advantage of the device's screen for real time previews of the videos or pictures being captured.

Two microphones. The use of more than one microphone augments the expressiveness of blow interaction. Moreover, the employment of two microphones placed in opposite directions enables the employment of noise reduction algorithms that may be very useful in order to improve audio interaction on its two dimensions: input and output.

Pressure sensitive multi-touch screen with haptic feedback. The abolishment of binary touch technology and its replacement with pressure sensitive touch technology envisions mostly an increase of interaction expressiveness. The same reason justifies the inclusion of multi-touch technology. On another strand, haptic feedback characteristics are included in order to mitigate the feedback loss, introduced by the substitution of the keypad by touch technology. Most touch screens available on commercialized hand-held devices eliminate the button feedback that users were familiar with in keypads. This introduces interaction issues, especially for visually impaired users that were suddenly inhibited of using such devices. The haptic feedback capabilities of our device's screen mitigate these issues, improving interaction for all kinds of users.

Pressure sensor array and matrix. Our device is literally wrapped in pressure sensitive technology. The pressure sensor arrays available on the device's laterals and the pressure sensor matrix available on its backside enable pressure sensitive grasp-awareness. Moreover, this technology enables the use of the sides and back of the device for pressure sensitive multi-touch input. For instance, by moving a finger on the side of the device or making gestures on its backside users may manipulate volume, scroll, zoom and other interaction parameters, without covering the screen with their hands.

Redundant context information. The prototype designed considers redundant context information. For instance, light-awareness is enabled by the available photo sensors and cameras. This redundancy may be employed in order to validate context information gathered by different sensing mechanisms that enable similar types of awareness. Moreover, it enables the use of the less energy-consuming sensing mechanisms, whenever validation is superfluous.

4 Conclusion and Future Work

In this paper we presented the initial design stages of a context aware multimodal hand-held device conceived in order to improve human-computer-interaction. The design started with an extensive requirement and feasibility analysis, which focused the contextual aspects we found necessary in order to achieve the goal proposed. This analysis included several experiments conducted in order to prove concepts that were not addressed on the literature. Afterwards, we presented the design of a prototype envisioned for this purpose and discussed the rationale of the hardware components employed.

Our future work relies on the construction of the proposed device. On the best case scenario this work will be sponsored by a well-known device manufacturer, which will be responsible for integrating the necessary hardware. On the worst case scenario, we will employ our studies and effort on the construction of a device integrating all the hardware mentioned, aiming at a proof of concept that integrates all the previously mentioned context information and interaction modalities. Some details (e.g. size, weight, battery life) were left out of this part of the research and will be focused on the continuity of the project here presented.

Acknowledgments. This work was supported by LaSIGE and FCT through the Multiannual Funding Programme and individual scholarships SFRH/BD/44433/2008.

References

1. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. In: Int. Journal of Ad Hoc and Ubiquitous Computing, Vol. 2, No.4, pages 263--277 (2000).
2. Abowd, D., Dey, A. K.: Towards a Better Understanding of Context and Context-Awareness. In: Procs. of the 1st Int. Symposium on Handheld and Ubiquitous Computing. Springer, Berlin, pp. 304--307 (1999).
3. Turk, M., Robertson, G.: Perceptual user interfaces introduction. Communications of the ACM 43, No. 3, pp. 33--35 (2000).
4. Baillie, L., Schatz, R.: Exploring multimodality in the laboratory and the field. In: Procs. of the 7th Int. Conference on Multimodal interfaces. Torento, Italy (2005).
5. Reis, T., Sá, M., Carriço, L.: Multimodal Interaction: Real Context Studies on Mobile Digital Artefacts. In Procs. of HAID, 3rd Int. Workshop on Haptic and Audio Interaction Design. LNCS 5270, Springer, Berlin, pp.60--69 (2008).
6. Oviatt, S., Darrell, T., Flickner, M.: Multimodal interfaces that flex, adapt, and persist. In: Communications of the ACM 47, No. 1, pp. 30--33 (2004).
7. Lai, J.: Facilitating Mobile Communication with Multimodal Access to Email Messages on a Cell Phone. In: Procs. of CHI'04. ACM Press, New York, pp. 1259--1262 (2004).
8. Want, R., Hopper, A., Falcao, V., Gibbons, J.: The Active Badge Location System. ACM Transactions on Information Systems 10(1), pp. 91--102 (1992).
9. Golding A., Lesh, N., Indoor Navigation Using a Diverse Set of Cheap Wearable Sensors. In: Procs. of the IEEE Int. Symposium on Wearable Computing, pp. 29--36 (1999).
10. Weiser, M.: The Computer of the 21st Century. Scientific American 265, 3, pp. 66—75 (1991).
11. Want, R., Schilit, B., Adams, N., Gold, R., Petersen, K., Goldberg, D., Ellis, J., Weiser, M.: The ParcTab Ubiquitous Computing Experiment. Tech. Report CSL-95-1, Xerox Palo Alto Research Center (1995).
12. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In: Procs. of the IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz. (1994).
13. Abowd, D., Atkeson, G., Hong, J., Long, S., Kooper, R., Pinkerton, M.: Cyberguide: a mobile context-aware tour guide. Wireless Networks, 3(5), pp.421—433 (1997).
14. Baber, C., Bristow, H., Cheng, S., Hedley, A., Kuriyama, Y., Lien, M., Pollard, J.,Sorrell, P.: Augmenting Museums and Art Galleries. In: Procs. of INTERACT ´01. (2001).
15. Dey, A., Abowd, D.: CyberMinder: A Context-Aware System for Supporting Reminders. In: LNCS Handheld and Ubiquitous Computing, Vol. 1927, pp. 201--207 (2000).
16. Saffo, P.: Sensors: The Next Wave of InfoTech Innovation. Ten-Year Forecast, Institute for the Future. (1997).

17. Darnauer, J., Garrity, S., Kim, T.: Orientation-based Interaction for Mobile Devices. Available at: http://hci.stanford.edu/~srk/cs377a-mobile/project/final/darnauer-garrity-kim.pdf
18. Gellersen, H., Schmidt, A., Beigl, M.: Multi-sensor context-awareness in mobile devices and smart artifacts. In: Mobile Networks and Applications, Vol.7, No.5, pp. 341--351 (2002).
19. Reis, T., Carriço, L., Duarte, C.: Mobile Interaction: Automatically Adapting Audio Output to Users and Contexts on Communication and Media Control Scenarios. To appear, accepted for publication on HCII (2009).
20. Kurasawa, H., Kawahara, Y., Morikawa, H., Aoyama T.: A Dynamic User Posture Inference Scheme for Mobile Devices. In: Procs. of the 8th Int. Conference on Ubiquitous Computing. Orange County, USA. (2006).
21. Yamabe, T., Takahashi, K.: Experiments in Mobile User Interface Adaptation for Walking Users. In Procs. of the Int. Conference on Intelligent Pervasive Computing. pp. 280--284 (2007).
22. Baek, J., Kim, S., Kim, H., Cho, J., Yun, B.: Recognition of User Activity for User Interface on a Mobile Device. In: Procs. of the 24th South East Asia Regional Computer Conference. Thailand (2007).
23. Brandon T. Taylor, B., Bove, V.: The Bar of Soap: A Grasp Recognition System Implemented. In: Procs. of Conference on Human Factors in Computing Systems. ACM 978-1-60558-012-8/08/04. Italy (2008).
24. Sun Sunspots. Available at: http://www.sunspotworld.com
25. Arduino. Available at: http://www.arduino.cc/
26. Hughes, S., O'Modhrain, S.: SHAKE - Sensor hardware accessory for kinesthetic expression Stephen Hughes. In: Procs. of 3rd International Conference on Enactive Interfaces. Ireland, pp. 155—156 (2006).
27. P. Klante, J. Kräosche, and S. Boll. Accesights - a multimodal location-aware mobile tourist information system. In Procs. of the ICCHP. (2004).
28. Fabian, H.; Gerald, B., Antje, D.: Egocentric Maps on Mobile Devices. In: Procs. of the 4th Int. Workshop on Mobile Computing. Stuttgart: Verlag, pp. 32--37 (2003).
29. Raento, M., Oulasvirta, A., Petit, R., Toivonen, H.,: ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications. IEEE Pervasive Computing, Vol. 4, No. 2, pp. 51-59 (2005).
30. Sun, G., Gu, Q.: Accelerometer based north finding system. In: Procs. of Position Location and Navigation Symposium. IEEE. USA. ISBN: 0-7803-5872-4 pp. 399--403 (2000).
31. Luley, P., Paletta, L., Almer, A., Schardt, M., Ringert, J.: Geo-Services and Computer Vision for Object Awareness in Mobile System Applications. In: LNGC 1863-2246. ISBN: 978-3-540-36727-7. Berlin, pp. 291--300 (2007).
32. Brown, J.: The Stick-e Document: a Framework for Creating Context-Aware Applications. Electronic Publishing '95 pp. 259-272 (1995).
33. Reimann, C., Paelke, V.: Computer Vision based Interaction Techniques for mobile Augmented Reality. In: 5th Paderborn Workshop Augmented and Virtual Reality. HNI-Verlagsschriftenreihe, No. 188. Germany (2006).
34. Reis, T., Carriço, L., Duarte, C.: Interaction Design: The Mobile Percussionist. To appear, accepted for publication on HAID (2009).
35. Simon, R., Kunczier, H., Anegg, H.: Towards Orientation-Aware Location Based Mobile Services. In: 3rd Symposium on LBS and TeleCartography. Austria (2005).
36. Mitchell, C.: Mobile Apps: Adjust Your Ring Volume For Ambient Noise. In: MSDN Magazine, available at: http://msdn.microsoft.com/en-us/magazine/cc163341.aspx (2008).
37. Sunlight illuminated and sunlight readable mobile phone. Patent available at: http://www.faqs.org/patents/app/20090061945

Sessão 2C: Processamento de Dados, Informação e Linguagem

Data Access Pattern Analysis based on Bayesian Updating

Stoyan Garbatov, João Cachopo, João Pereira

Instituto de Engenharia de Sistemas e Computadores Investigação e Desenvolvimento,
R. Alves Redol 9, 1000, Lisboa, Portugal
stoyangarbatov@gmail.com, joao.cachopo@inesc-id.pt, joao@inesc-id.pt

Abstract. A new system is presented for analyzing and predicting the most common data access patterns performed during the execution of applications. The correct and precise functioning of the system is demonstrated though the execution of the OO7 benchmark modeled with Monte Carlo simulations. The results obtained show that the precision presented by the system is high. The overheads introduced by the systems are in the order of 4%-5% slower, with respect to the non-instrumented application. The system is flexible enough to be applied to any application where there is a need to identify the access patterns performed.

Keywords: data access pattern, Monte Carlo, Bayesian updating.

1. Introduction

To perform their functions, most applications need to fetch data from external resources (e.g., from files on disk, from a database, or from other applications in a distributed system), which is a costly operation. Because of this high cost in fetching data, applications that access large volumes of data must be carefully engineered to have an acceptable performance.

The cost of fetching data from an external resource depends on a series of factors, but, often, a good design rule is to minimize the number of round-trips made by the application to load the data. On the other hand, we want to minimize the amount of data fetched. Thus, ideally, to perform a given operation, we want to do a single round-trip that fetches exactly the data that we need for the operation. A well-known example of such optimized data-fetching approaches is the use of complex SQL queries to retrieve data from multiple tables at once from a relational database.

Unfortunately, this ideal goal is seldom possible to achieve and is becoming harder to come close to. The reasons are manifold.

The current state-of-the-practice is to leave to the programmers the burden of knowing what to fetch and when to fetch it. However, as applications become more complex and are developed with higher-level languages (such as object-oriented programming languages), knowing beforehand which data is needed for a particular computation becomes humanly unfeasible.

Moreover, even if the data-fetching patterns for an application are fine-tuned, these adjustments may become useless or even counterproductive when either the requirements for the application or the data structure change, even if slightly.

To complicate things further, many applications use caches to reduce the need to fetch data from external resources, and, thereby, accelerate the application. The use of these caches and their dynamic contents influence the optimal fetching strategy.

Therefore, we argue that the data-fetching strategy for an application should not be a responsibility of the programmers. Rather, it should be determined automatically, based on the data-access patterns that are dynamically observed for the application.

In this paper, we concentrate on the problem of determining what are the data access patterns of an application. We describe the design and implementation of a system that predicts the data-access patterns for a Java application. To do that, the system captures all of the data-accesses made by the application during its execution, and uses statistical analysis over the captured data to predict future data-accesses.

Our system was designed to integrate seamlessly with the development of a Java application, and requires minimal effort from the programmer. Moreover, our system is efficient both in the amount of memory that it requires and in the performance overheads that it introduces, so that it may be used on production environments.

In the following Section, we describe in greater detail the kind of applications that our system was designed to work with. Then, in Section 3, we describe the design and implementation of our system, both in what regards the capturing of data-accesses and their analysis. In Section 4, we evaluate how our system performs on a well-known benchmark. In Section 5, we discuss related work. Finally, in Section 6, we present the major contributions and conclusions of this work.

2. Characterizing applications and data-access patterns

Applications may access data and be developed in many different ways. Thus, even though the techniques that we describe in this paper may have a broader applicability, for the purposes of this presentation, we concentrate only on a certain class of applications: object-oriented applications that access data through a domain model. More specifically, we assume that applications compile to the Java Virtual Machine, and that they represent the externally accessible data via a set of classes and their corresponding fields. We chose to concentrate on this set of applications because it includes a significant portion of the modern enterprise applications.

So, given such an application, what we need to understand now, is what its data-access patterns are. Informally, the data-access patterns of an application tell us which data is more often used by the application for each of its operations. Yet, this informal definition may range from very fine-grained information (such as knowing that when executing an operation O with arguments A, the application needs to read the slot S of instance I of a class C), to more coarse-grained information (such as knowing that when executing the operation O, regardless of its arguments, the application needs to read some data from class C with a given probability and some data from class C2 with another probability), with a lot of variations in between.

Choosing one solution in this space represents a tradeoff. Using fine-grained

information has the potential of being more accurate for a given situation that was seen before, but on the other hand it may be more difficult to predict what to do in situations never seen before (e.g., a call to operation O with arguments B). Moreover, storing and using such fine-grained information will necessarily introduce higher overheads into the application.

The system that we propose uses a more coarse-grained approach, aligned with the other end of the space. A data-access pattern in our system describes which classes and which fields within those classes are accessed, and with which probability, for each possible context of the application. We describe these concepts in greater detail in the following Section.

3. Description of the system

Our system is composed of two modules: the data-acquisition module and the data-analysis module. The data-acquisition module is responsible for collecting the information about the data-access patterns during the execution of an application. The data-analysis module is responsible for applying the Bayesian-updating model to the collected data and, subsequently, for generating predictions about the future access patterns. Each of these modules shall be discussed with greater detail in the following sections.

3.1 Data Acquisition

As we saw, the data-access patterns for an application describe which kind of domain objects and which of their fields are accessed during the execution of the application. So, to capture these patterns, we need to identify all the points where data is accessed within the application and instrument them to register those accesses during the execution of the application.

Our system performs this instrumentation at compile-time via bytecode rewriting. Thus, the system performs all the necessary modifications to the target application in an automatic way, without the intervention of the programmer. The modifications are achieved through two instrumentation phases, both of which make use of the *Javassist*[1] library to inject the code.

The first instrumentation phase injects the code necessary for the identification of the *contexts* within which all data-accesses take place. The *context* is a key concept in our system. It corresponds to the sequence of method invocations that precede a given point in the execution of the application and define the surrounding scope for each data-access. The actual sequence of methods to take into account when defining a context depends on the *context depth*. This can be better explained through the following example code:

[1] *Javassist* is a class library for editing bytecodes in Java.

```
public void methodA() {
   //method body
   methodB();
   //method body
}
public void methodB() {
   //method body
   methodC();
   //method body
}
public void methodC(){
   //method body
}
```

Assume that methods *methodB* and *methodC* are invoked only as shown above. The *context* of any field access made in the body of *methodC* will be defined by the sequence <*methodA*, *methodB*, *methodC*>, if the context depth is 3, or by the sequence <*methodB*, *methodC*>, if the context depth is 2. The context depth is a parameter that may be adjusted when is necessary to alter the granularity of the input data that is accumulated while the application is in execution (this is the data that will be used for the probabilistic analysis). The context depth can take values from 1 to any deemed practical.

To identify the contexts at runtime, the first instrumentation-phase modifies each method of the application. It injects code that updates the context information upon entering the method, and code that cleans the context upon returning from it.

The second instrumentation-phase replaces every field access that exists within the application with the invocation of a previously injected statically typed method. There is a distinct method for each of the fields for every class of the application. These methods determine the surrounding context in which the field is accessed and update the associated statistical information. This is done according to the type of access performed: a distinct method is invoked whether the field is read or written.

Once these two phases are complete, the application can be put into operation, and it will automatically collect the statistics about each data-access, without any further modifications.

An important aspect of the solution presented here are the data structures used to store the statistical data. During the instrumentation phases, the system performs an analysis of the structure of the target application via reflection. As a result of this analysis, it generates a set of *PClass* instances to model the structure of each of the target application classes. Each of these *PClass* instances contains a set of *PField* instances, which are used to represent each of the fields that exist in the application class. The information kept in a *PField* covers not only the name and type of a field, but also the number of times it has been accessed in a context. The subsequent probabilistic analysis uses this information to make its calculations.

The relationship between *PClass* instances and an application class is many-to-one. This can be explained with the fact that the *PClass* instances store information regarding the way that application classes are accessed in the contexts during execution. Consequently, if the same class is used in different contexts, distinct *PClass* instances will keep the information about how that class was used in each.

Taking this into account, it is possible to estimate the upper bounds on the memory requirements for maintaining these structures. The memory will be proportional to the

number of classes, times their average number of fields, times the average number of distinct contexts where each class is accessed during the execution. More importantly, note that the memory requirements will not grow continuously, independently of the duration of the execution of the application. In general, this consumption of memory is negligible when compared to the memory needed by the application.

Moreover, given that the probabilistic analysis iterates through the *PField* and *PClass* instances, the time spent in the probabilistic analysis will also take time that is proportional to the previously stated bound.

3.2 Data Analysis

Once the necessary information has been gathered, we may make predictions about the application's data-access patterns. These predictions result from a probabilistic analysis based on Bayesian techniques. An initial study of this work has been presented in [1] and due to space limitations, the details regarding the formulae used to obtain the final results are omitted, but they may be found in an extended version of the paper in [2].

Bayesian analysis techniques are used for parameter estimation. They give an estimate of the statistical uncertainty of the estimated parameters (corresponding, in this case, to the likelihood of reading/writing a given field of an application class) and can update them when new information becomes available. This is done using two sets of data. The first set is designed by *prior* and corresponds to the data accumulated up to a certain point in the past. The second set is called *current* and is defined by the data acquired from that point in the past, up to the current moment. Based on these two sets, a *posterior* set is calculated. Once all these three sets are available, the prediction for the future access patterns can be performed.

The first phase of the analysis is to generate the *prior* and *current* histograms for the data, which can be seen in Fig. 1. The data on which they are based corresponds to the number of times every field was accessed in each of the identified contexts, during the execution of the application. The x axis of the histogram corresponds to the classes existing in the relative weight of each field access, on a global level. There are 10 distinct classes. The value used to identify each class corresponds to the middle of the class. The y axis corresponds to the number (frequency) of fields whose observed probability of being accessed belongs to a given class.

It is important to point out that it is assumed that the distribution functions that originate from the *prior* and *posterior* histograms share the same type (e.g. normal distribution), a so-called *conjugated prior*. The statistical descriptors[2] for the *prior*, *current*, and *posterior* are determined. In the literature, a number of prior, posterior, and predictive distribution functions can be found, see [3-5].

The continuous normal probability density function is presented by a histogram having the same statistical descriptors. The predicted histogram is adjusted to the statistical descriptors of the predicted normal probability density function by applying a special numerical procedure to make the mean and the standard deviation of the histogram equal to the continuous distribution.

[2] The statistical descriptors correspond to the mean value and variance.

Based on the *prior*, *current*, and *posterior* statistical descriptors, the *predicted* histogram is generated. There is an optimization process involved to calculate the precise *predicted* distribution that adapts with the minimum error with respect to the expected statistical descriptors that were calculated by the Bayesian approach.

4. Experimental results and Evaluation of the System

We used the OO7 benchmark [6] to evaluate our system. We chose this benchmark because it presents itself as a data intensive application that performs complex manipulations. The OO7 benchmark was built to evaluate the performance of object-oriented persistence mechanisms for an idealized CAD (computer aided design) application. The benchmark performs various types of traversals and read/write operations (grouped into 14 main methods) over its domain objects, which are organized into an elaborate hierarchy.

The benchmark is completely deterministic and to effectively validate the new probabilistic method that we propose in this work, a stochastic[3] application should be used. Therefore, we modeled the number of invocations of the main methods of the benchmark through Monte Carlo simulations. By doing that, the benchmark is converted to a stochastic model.

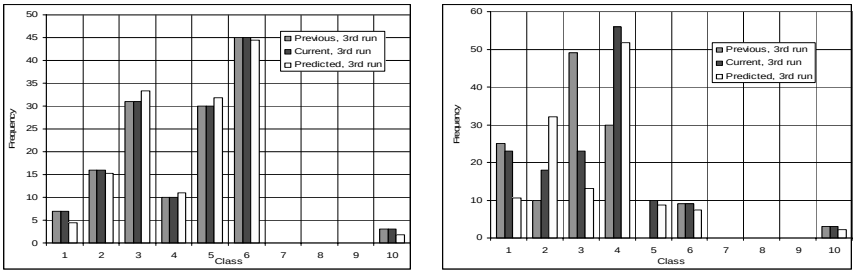


Fig. 1. Result histograms, left and right mode location input

We used uniform random numbers to generate the Monte Carlo simulations and the obtained output is a triangular distribution that represents the uncertainty in the calls of different methods.

Monte Carlo simulation is categorized as a sampling method because the inputs are randomly generated from probability distributions to simulate the process of sampling from an actual population. So, a distribution is chosen for the inputs, that most closely matches the current state of knowledge. The data generated from the simulation can be represented as probability distributions (or histograms).

The triangular distribution is typically used as a subjective description of a population for which there is only limited sample data. It is based on knowledge of the minimum and maximum and an inspired guess as to what the modal value might

[3] A stochastic process is one whose behavior is non-deterministic in that a system's subsequent state is determined both by the process's predictable actions and by a random element.

be. Despite being a simplistic description of a population, it is a very useful distribution for modeling processes where the relationship between variables is known, but data is scarce, because of the high cost of collection.

Three different triangular distributions for the call of the fourteen main methods of the benchmark were generated, based on Monte Carlo simulations. They are left, right, and middle mode location.

Random number generation for a triangular distribution is performed by transforming a continual uniform variable into the range 0 to 1 with the distribution's inverse probability function.

As described in the previous section, the Bayesian updating technique uses two sources of data to make a prediction. One of them is the *previous*, and the second one is the *current*. The *current* data is used to "update" the *previous* collected information, by making it reflect the patterns that have been most recently observed. By doing this, the *predicted* histogram is obtained, as can be seen in Fig. 1. The x axes of the histograms represent the index of the classes within which a field belongs. The index *i* is defined by $i = floor\left[(p + 0.1)/0.1\right]$ where $p \in [0,1]$. It should be noticed that, whereas the bars in the *previous* and *current* histograms indicate the number of fields whose observed probability of being read/written belongs to a given class, the *predicted* histogram reflects the fluctuation in the relative importance of each of the classes, which is expected to be seen in the following executions of the application, see textbook on Bayesian statistics, e.g. Box & Tiao [7] and Lindley [8].

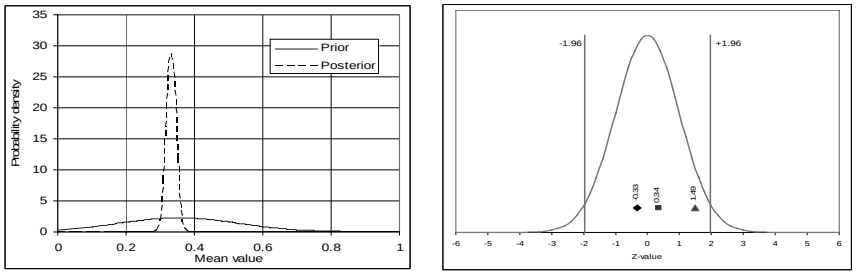


Fig. 2. *Prior* and *posterior* probability function (mid mode location) and respective predicted probability density function for the expected mean value (left), Z values test (right)

Based on the Bayesian method, both the physical uncertainty related to the considered variable as well as the statistical uncertainty related to the model parameters can be quantified. An important property of the Bayesian analysis is the fact that the uncertainty of the prediction is reduced. This can be seen in Fig. 2, left.

To evaluate the precision of the results generated by the system, we need to compare the predictions generated during the second runs of the benchmark, with the actual patterns observed during its third runs. To check whether the mean of the prediction is significantly different from the one observed in the next execution, we resorted to a null hypothesis test. The Z test statistic for this type of check can be found in [7] as:

$$z = \mu_1 - \mu_2 \Big/ \sqrt{\sigma_1^2/n_1 + \sigma_2^2/n_2} \qquad \textbf{(1)}$$

where μ_1 and μ_2 are the mean values, σ_1^2 and σ_2^2 are the variances and n_1 and n_2 are the sample sizes, for the predicted and the observed, respectively.

The critical value for z is defined as ± 1.96, that corresponds to 0.05 level of significance to reject the null hypothesis.

For the three different input modes tested here, the calculated z values are presented in Fig. 2, right. As can be seen, all three z values belong to the area of 95% confidence that the mean values of the predicted and subsequently observed are not different. Consequently, we may conclude that the predictions created by the system are precise enough. They are able to indicate correctly the tendencies that are actually observed in the next executions of the application.

As has been previously referred to, there is a significant amount of code injected in the application. The need to execute this code causes overheads and penalizes the performance of the application, in comparison with its non instrumented version. So, it is important to measure those overheads. The execution times of the main methods of the OO7 benchmark used for the results are summarized in Fig. 3.

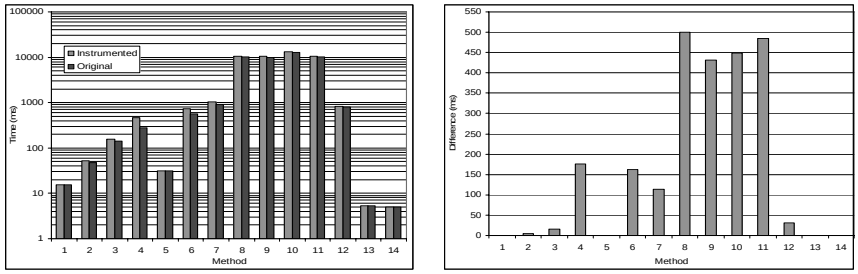


Fig. 3. Execution times (logarithmic scale), left, difference between execution times, right

In the x-axis of the histogram presented in Fig. 3 we have each of the 14 main methods that define the benchmark, and the values shown are in milliseconds. The weighted average of the overhead equals 5.15%. As a result of that, the instrumented version is, on average, about 5% slower, in its execution. It is deemed that the performance penalty is acceptable.

5. Related Work

Recently, many related works have been reported in the literature. Knafla presented in [9] a methodology where the main goal is to facilitate the pre-fetching of persistent objects through the prediction of data access.

The relationships between the objects are modeled by a discrete-time Markov Chain. A discrete-time Markov Chain is a stochastic process through which it is possible to model the behavior of systems. The model is constructed in such a way that the following state to which the system is going to transit does not depend on any previous states through which the system has already passed (but only on the current).

The work above offers an interesting view regarding the use of stochastic models for predicting the behavior of applications. However, the method is stateless and, consequently, cannot make use of any previous information that may be available.

Han et al presented in [10], a new technique for pre-fetching. One of the fundamental ideas behind the work is the fact that the access patterns employed by applications can be modeled in terms of the attribute references made when manipulating objects instead of the pattern of object references.

Bernstein et al. presented in [11] a technique for data pre-fetching. The main concept behind it is to associate a context to every object at the time it is loaded. This subsequently allows using the context of the object and the concrete portion of an object's state that is loaded to interpolate about the probability of the application needing to manipulate the same portion of state of other objects sharing that context. That enables the loading of data that would be needed by the application in a pre-emptive fashion (pre-fetching it), effectively reducing the time lost while waiting for the data to be loaded on demand. Other interesting works, with respect to efficient persistent data manipulations, can be found in [12-16].

The work presented here is a part of a project which deals with optimization techniques, such as pre-fetching or caching. In order to achieve this, it is necessary to know the access patterns performed. Most of the works mentioned above, do not apply any specialized analysis techniques. The main contribution of this work resides in the development of a new methodology for identifying access patterns based on high level statistical techniques.

6. Conclusions

In this paper, a new system for analyzing the most common data access patterns performed during the extensive operation of an application was presented. The data accesses are analyzed and predicted using advanced probabilistic techniques employing Bayesian Updating. Several scenarios were generated and executed, and the subsequent results were analyzed. The system developed for data access pattern analysis was applied over the OO7 benchmark adapted to make use of Monte Carlo simulation.

The overheads introduced by the system were shown to be in the order of 4-5%, when comparing the instrumented application with the original one. For the three different input modes tested here, the calculated z values belong to the area of 95% confidence that the mean values of the predicted and subsequently observed are not different. Consequently, we may conclude that the predictions created by the system are precise enough. All the initially identified requirements were satisfied and the objectives were achieved.

The system developed here is flexible enough to be used as a basis for a set of optimization techniques. Special benefits could be obtained when the data being analyzed is persistent. This would allow for the application of optimizations regarding the way that the information is loaded (e.g. pre-fetching), is stored (e.g. multiple data-bases store the application data) or is cached (caching policies).

Acknowledgments

This work has been performed in the scope of the Pastramy project (PTDC/EIA/72405/2006), which is funded by the Portuguese FCT (Fundação para a Ciência e a Tecnologia).

References

1. Garbatov, S. and Cachopo, J., CoPO: Collections of Persistent Objects, INESC-ID Tec. Rep. 9/2009.
2. Garbatov, S., Cachopo, J. and Pereira, J., Data Access Pattern Analysis based on Bayesian Updating, Tec. Rep. 2009.
3. Raiffa, H. and Schlaifer, R.: Applied Statistical Decision Theory. MIT Press, Cambridge, (1968).
4. Aitchison, J. and Dunsmore, I.R.: Statistical Prediction Analysis. Cambridge University Press, Cambridge, (1975).
5. Rackwitz, R. and Schrupp, K.: Quality Control, Proof Testing and Structural Reliability. Structural Safety, Vol. 2, pp. 239-244, (1985).
6. Carey, M.J., Dewitt, D.J. and Naughton, J.F.: The oo7 benchmark, pp. 12–21, (1993).
7. Box, G. E. P. and Tiao, G. C.: Bayesian Inference in Statistical Analysis, Wiley, New York, (1992).
8. Lindley, D.V.: Introduction to Probability and Statistics from a Bayesian Viewpoint, Vol. 1+2, Cambridge Univ. Press, Cambridge, (1976).
9. Knafla, N.: Analysing object relationships to predict page access for prefetching. In Proc. of the Eighth Int. Workshop on Persistent Object Systems: Design, Implementation and Use (POS-8). pp. 160–170, (1998).
10. Han, W. S., Whang, K.Y. and Moon, Y. S.: Prefetching based on the type-level access pattern in object-relational dbmss. In Proc. of the 25th Very Large Data Base Conference (VLDB99), (1999).
11. Bernstein, P. A., Pal, S. and Shutt, D.: Context-based prefetch for implementing objects on relations. In Proc. of the 25th Very Large Data Base Conference (VLDB99), (1999).
12. Willis, D., Pearce, D. J. and Noble, J.: Efficient object querying for java. In: In Proc. of the European Conference on Object-Oriented Programming (ECOOP), (2006).
13. Willis, D., Pearce, D. J. and Noble, J.: Caching and incrementalisation in the java query language. In Proc. of the Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA08), (2008).
14. Ibrahim, A. and Cook, W. R.: Automatic prefetching by traversal profiling in object persistence architectures. In Proc. of the European Conference on Object-Oriented Programming (ECOOP), (2006).
15. Wiedermann, B. and Cook, W. R.: Extracting queries by static analysis of transparent persistence. In: In Proc. of the 34th Symposium on Principles of Programming Languages (POPL07), (2007), 199–210.
16. Wiedermann, B., Ibrahim, A. and Cook, W.R.: Interprocedural query extraction for transparent persistence. In: In Proc. of the Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA08), (2008).

Automated Social Network Epidemic Data Collector

Luis F Lopes, João M Zamite, Bruno C Tavares, Francisco M Couto, Fabrício Silva, and Mário J Silva

LaSIGE, Universidade de Lisboa
epiwork@di.fc.ul.pt

Abstract. Recent epidemiological surveillance projects began collecting data from the Internet to identify inflectional diseases propagation. These systems collect data from pre-selected data sources somehow related to the subject. However, other sources, like web social networks, may present early evidences of an infection event. With the increasing popularity of social networks, where people post personal data interactively, we may hint the outbreak of an epidemic event from these data. This paper presents the architecture of a system capable of collecting epidemiological data from social network services. A preliminary prototype collects data from Twitter into a local database, where it can be analyzed and made available to other applications. Initial evaluation results show that relevant epidemiological data can be found in this source.

Key words: Epidemiological surveillance, Data Retrieval, Social networks, Twitter

1 Introduction

Epidemiological surveillance systems are essential for the study and control of diseases. Data recovered by these systems should be as extensive as possible in order to obtain enough information to understand the propagation of diseases, assessing their impact in public health through epidemiological prediction tools.

In recent years there has been an increase of quantitative social, demographic and behavioral data available, which can be used by statistical and mathematical models to improve the traditional disease surveillance systems, providing faster and better geo-referenced outbreak detection capacities.

The epidemiological surveillance is traditionally done by governmental and international health organizations, such as the WHO (World Health Organization), and is mostly based on cases reported by health services.

Although official statistics should be accurate and based on reliable data, new technologies can be implemented to collect epidemiological data that could be regarded in the future as a valuable complement to national reporting systems.

The use of the web to collect information about epidemics has been a target of investigation in the last years. State-of-the-art systems can collect data from

different kinds of sources, such as search-engine log data, news sites and user-provided input. However, social networking sites are potential targets for the retrieval of epidemic information, given their role as forums where people meet and post information about themselves. Text messages exchanged on the web can be used to identify cases of diseases or at least to provide an idea of the spread of a disease in a community.

Twitter[1] functions as a microblog, where people post small messages with a limit of 140 characters that can either be visualized by anyone or by specific friends, according to the account settings. The large and still increasing number of users of Twitter makes it the perfect case case study for our approach.

Data obtained from social network services such as Twitter, can provide real time information from a large base of regular users. Furthermore, since this information is available instantaneously, it can be used for early detection comparatively to information collected from official sources as suggested by Google Trends observations [3].

This paper presents the Data Collector that is being developed as a module of the Epiwork's Epidemic Marketplace[2], an epidemiological data management platform. The objective of this module is to explore the use of web based social network services for the collection of epidemiological related data. This module collects data from Twitter, searching for disease keywords in the published messages. Besides the text of the message, the module retrieves information about the author, location and posting date. In this work we present some statistics related to the data collector and analyze the data obtained for H1N1 in some European countries and discuss how to improve the system.

2 Related Work

Some systems have been developed in the last years to collect disease information from Internet users. Internet monitoring systems (IMS) use data obtained from user voluntary reports, such as Gripenet [6] or collected automatically, like Google Flu Trends [3] and HealthMap [1].

An IMS is an information collection system that depends on the active participation of registered users, which receive weekly newsletters about the flu and are invited to fill a questionnaire about the flu symptoms (or their absence in the previous week). Gripenet was developed after the model of Holland's Influenzanet [5]. It has been implemented until now in 6 countries, besides Portugal and Holland: Belgium, Italy[3], Brasil[4], Mexico[5], United Kingdom[6] and Australia[7].

[1] http://www.twitter.com/
[2] http://epiwork.di.fc.ul.pt/
[3] http://www.influweb.it/
[4] http://www.gripenet.com.br/
[5] http://reporta.c3.org.mx/
[6] http://www.flusurvey.org.uk/
[7] http://www.flutracking.net/

More should be expected to appear in the future with the objective of having real time flu monitorization.

Healthmap[1] is a website that displays in the world map information about new cases of diseases, especially infectious diseases, collected from several sources. These include news sources, such as Internet news sites, the ProMED-mail newsletter[4], Eurosurveillance and WHO. The degree of reliability of these sources is variable, ranging from media news to validated official alerts. The data displayed is organized by disease using an automated text processing system and then displayed on the Earth map. The original information is available by a hyperlink to the original source.

Google Flu Trends has been recently announced as a system capable of predicting influenza epidemics based on search engine query log data [3]. It was initially deployed for the United States of America, where about 90 million users are believed to search online for information about specific diseases. It has now been applied in Australia New Zealand and experimentally to Mexico. The authors of this work state that certain queries can be used as a surveillance tool to predict the number of influenza cases and that these predictions can be produced one to two weeks before the official data.

In recent years Web based social network services, where people share their interests and activities, have become a popular trend. Moreover the types of information shared in these Social Networks are varied, from link sharing to short text messages. This variety allowed for News Services to gather and publish information in these networks.

3 Data Collector

The Data Collector is able to actively collect information about putative infections by automatically retrieving infection alerts from the web. The data retrieved is stored in a local database and made available through web services and a web user interface.

3.1 Architecture

The data collector (fig. 1) is composed by a backend and a frontend. The backend contains the web crawler which retrieves messages from Twitter, a relational database used to store the collected data and web services which can be used to access this data. The frontend consists of a web interface that will provide a dynamic graphic environment for the user to explore the data.

3.2 Data Storage

All the information retrieved from the web will be stored in a local database. Data is stored according to the UML class diagram shown in fig. 2. The two main classes are Disease and Location, which contain the diseases and the geographical locations to monitor. Each disease will have an official name and a type;

alternative names are instances of the Disease Alias class. In the same way, each location will have an official name and a type; alternative names are instances of the Location Alias class. The type of a location can be a city, country, etc. Each location will be geo-referenced by the latitude and longitude of its centre and a radius limiting the area to monitor. For example, cities will tend to have smaller radius than countries.

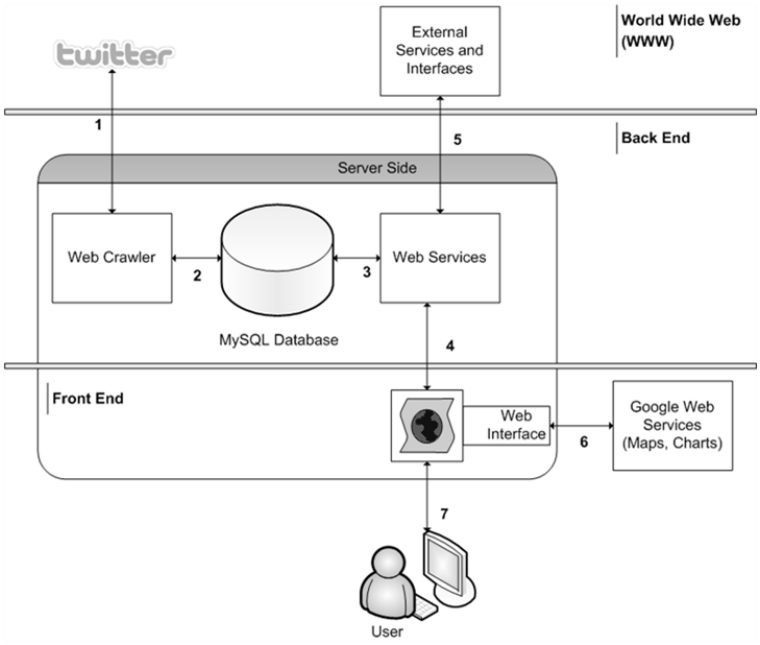


Fig. 1. Architecture of the Data Collector. 1 - The Web Crawler requests XML messages from twitter containing a disease and a location; 2 - The Web Crawler stores detected occurrences in a relational database; 3 - Web Services query the database for information; 4 - The Interface uses the Web Services to present data using 6 - other available Web Services to 7 - display the information to user in an interactive manner. 5 - Other external parties can use the web services for their own purposes.

A relationship between a location and a disease is stored when an alert is detected for the given disease at the given location. The Occurrence will store the date, author, source, evidence, type and score. The author, the source and evidence are information that may be retrieved or received attached to the occurrence alert. The attribute score will store a confidence score for each detected occurrence allowing the weighting according to specific attributes, once mathematical models are developed.

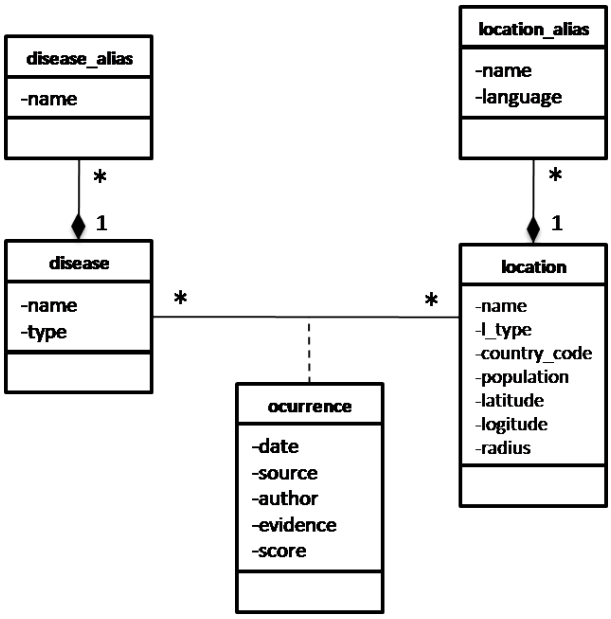


Fig. 2. UML class diagram of the relational database.

3.3 Web Crawler

This module actively collects information about putative infections by automatically crawling the social web. The current crawler accesses the Twitter using the web services documented in the Twitter Search API. To detect disease referencing tweets it queries for the name of a disease and of a location. For example, using: http://search.twitter.com/search.atom?lang=enq=h1n1+italy (see fig. 3).

3.4 Application Programming Interface (API)

The Data Collector provides a RESTful API which specifies the methods and parameters through which other users and applications can access the data contained in the database.

There are three main divisions in the API which specify methods to retrieve locations, diseases and occurrences in the database, displayed in HTML tables.

Retrieving information on diseases is done through the "diseases" method[8] which returns the names and all the aliases of each disease in our database. Requesting information on locations is done through the "locations" method[9] which returns all the information on the locations in our database.

[8] http://epiwork.di.fc.ul.pt/collector/diseases
[9] http://epiwork.di.fc.ul.pt/collector/locations

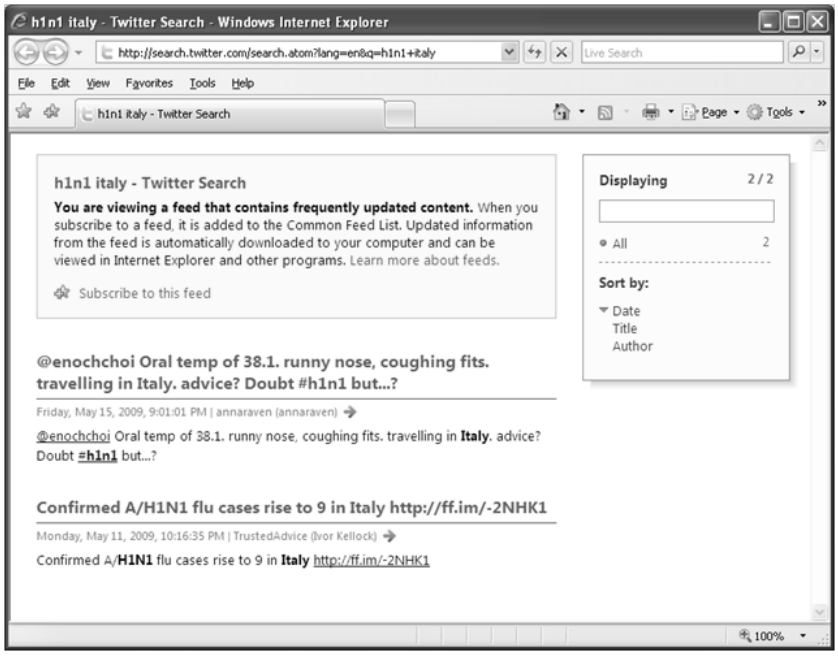


Fig. 3. Output of Twitter API when searching for h1n1 and Italy.

The "occurrence" method[10] returns the main name of the detected diseases, the main name of the location where they were detected as well as the source, author, evidence, score and date. This method can be filtered by location and by disease names using the "location=" and the "disease=" parameters. It can also be further filtered by date, specifying the "before=" and "after=" parameters in the format: "yyyyMMDDhhmmss". The "occurrence" method can also be used with the parameter 'format=tsv' which results in the method returning a dataset in tab-separated values (TSV) format. An example of the usage of this web service would be:

http://epiwork.di.fc.ul.pt/collector/occurrences?disease=h1n1&location= portugal&before=20090518000000&after=20090515000000&format=tsv

This request would produce a TSV containing the attributes of occurrences of H1N1 in Portugal, in a specific time interval.

3.5 Demo Mashup Client

The web interface (see fig. 4) will use AJAX technology to produce dynamic trends graphs and geographical maps that change in real time. It will permit the user to select and visualize data within a specific timeframe (i.e. within a week,

[10] http://epiwork.di.fc.ul.pt/collector/occurrences

month, year, etc), allowing for easier visualization of trends that occur in that window of time. The user interface is currently under development.

For this purpose, we intend to implement dynamical plots, using Google Chart[11], which will allow the user to visualize occurrences over time. Another means of visualizing data would be an earth map, using Google Maps[12]. The map will display markers color-coded according to the number of occurrences which will give the user immediate insight into the disease distribution. The displayed information could be interactively filtered by disease, location and source.

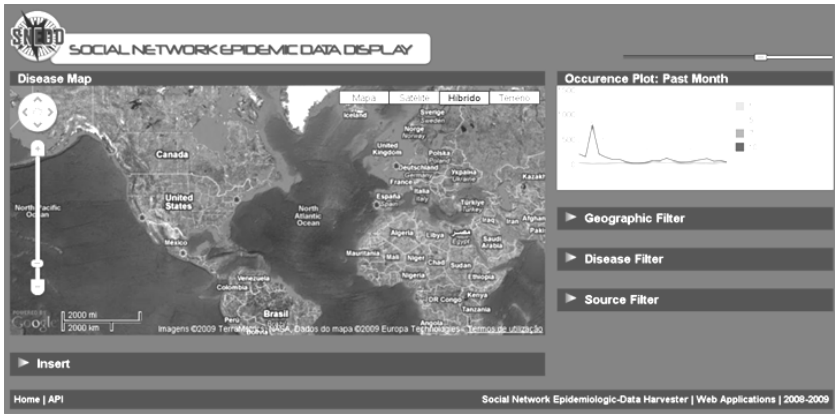


Fig. 4. Snapshot of the mashup webpage.

4 Implementation

The Data Collector is being implemented with free open source software. The collected data is currently being stored in a relational database implemented with MySQL Community Server 5.0. The crawler was written in PHP. It queries Twitter using its Search API for messages containing location names and disease names. It does so for all the possible combinations of locations and diseases. Location information about the countries and their capitals was retrieved from the geonames[13] webservice.Using the searchJSON method we retrieved the countryCodes and geonameId of all the countries of the world and their capitals[14][15]. The country area was retrieved using the CountryInfoCSV method[16]

[11] http://code.google.com/intl/pt-PT/apis/chart/
[12] http://code.google.com/intl/pt-PT/apis/maps/
[13] http://www.geonames.org
[14] http://ws.geonames.org/searchJSON?featureCode=PCLI
[15] http://ws.geonames.org/searchJSON?featureCode=PPLC&country=<countryCode>
[16] http://ws.geonames.org/countryInfoCSV?country=<countryCode>

which was used for calculating the radius of interest for each country. Using the geonameId the remaining information, including all the aliases was retrieved using the getJSON method[17]. Diseases were retrieved manually from the United States CDC website[18] and Health Protection Agency website[19] as well as other sources, including Healthmap. We are currently tracking 89 diseases as well as all the countries and their capitals (17165 names being used). The process of querying Twitter for all this information currently takes from 2 to 3 days, which we consider adequate to the system's needs considering the large number of location names being used. A new querying process is started each week, and since twitter stores messages for over 20 days there are no gaps or discrepancies on the messages obtainable each day.

5 Results

The crawler has been collecting information from tweets that contain a disease and a location in the message body. This type of search is still very simple and is probably missing lots of tweets. However, even so the system is collecting on average 3200 messages every day, from which 700 pertain to H1N1. Using the Data Collector web services, we produced datasets containing the data collected for the disease term H1N1 for several countries: Portugal, Spain, France, United Kingdom (UK), Holland and Italy. From the datasets the number of daily messages was analyzed (see fig. 5).

We observed that France and the UK are, among the countries analyzed, the ones where there were more references to H1N1 (or the alias swine flu). Even using a naïve search method it is possible to obtain a large number of messages. It is also possible to observe an increase in the number of messages being collected, which is correlated to the increase of the number of cases.

6 Conclusions

This work shows that it is possible to collect large amounts of epidemiological data from twitter, a system with a large base of users and daily messages.

Despite the large number of messages already collected by this system, it can be improved to collect even more information in a more accurate way. Since the first implementation the number of diseases and locations has already been largely increased. The system is now looking for a large number of locations and its aliases, based on data retrieved from Geonames. Although the number of diseases being currently tracked is already very high (89 diseases), the number of aliases used is still low. In the future this will be improved by implementing the UMLS ontology [2], which will increase disease coverage.

[17] http://ws.geonames.org/getJSON?geonameId=<geonameId>
[18] http://www.cdc.gov/ncphi/disss/nndss/phs/infdis2007.htm
[19] http://www.hpa.org.uk/

These messages that have been collected had the country in the text body while in the future it will recover messages that were posted in those countries but do not necessarily have the country name in the message.

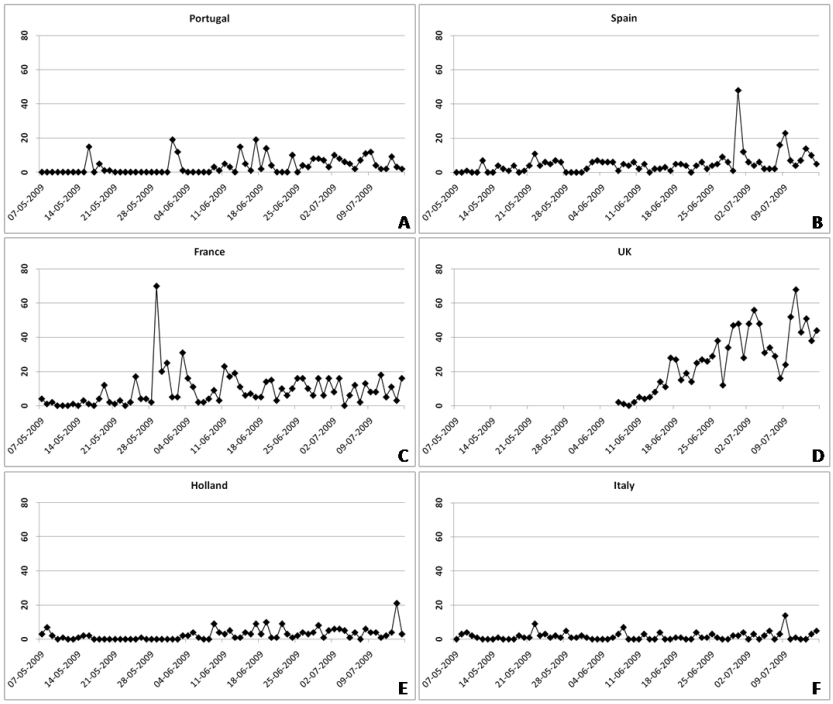


Fig. 5. Number of daily messages containing the word H1N1, from the 7th of May until the 15th of July of 2009, in six countries: A- Portugal, B- Spain, C- France, D- United Kingdom (only data from the 7th of June to 15th of July), E- Holland and F- Italy.

The hypothesis we intend to test is that the number of messages posted on Twitter related to a specific disease should be related to the number of disease cases in the population. Although there are several detected cases where collected messages do not reflect specific cases, we suppose that the amount of data is so large that those cases will not influence the results significantly,as observed by Google Trends [3].

The presented system will surely be more efficient in some countries, where the number of people using Twitter or other web based social network services is comparatively high. However this effect may be corrected through normalization or other bias elimination methods.

In the future the use of better text mining techniques will enable filtering the messages containing non relevant messages, improving the data accuracy.

Further statistical analysis and data collection is also required to validate the usage of data collected by this system in epidemiological propagation predictions.

7 Acknowledgements

The authors want to thank the European Commission for the financial support of the Epiwork project under the Seventh Framework Programme (Grant # 231807), the Epiwork project partners and FCT (Portuguese research funding agency) for its LASIGE Multi-annual support.

References

1. Brownstein JS, Freifeld CC. (2007). HealthMap: the development of automated real-time internet surveillance for epidemic intelligence. Euro Surveill, 12(11): E071129.5.
2. Bodenreider, O. (2004). The Unified Medical Language System (UMLS): integrating biomedical terminology. Nucleic Acids Research 32(Database issue):D267-270.
3. Ginsberg J, Mohebbi MH, Patel RS, Brammer L, Smolinski MS, Brilliant L. (2008). Detecting influenza epidemics using search engine query data. Nature, 457, 1012-1014.
4. Madoff, L. C. (2004). ProMED - mail: An Early Warning System for Emerging Diseases. Clinical Infectious Diseases, 39:227-232.
5. Marquet RL, Bartelds AI, van Noort SP, Koppeschaar CE, Paget J, Schellevis FG, van der Zee J. (2006). Internet-based monitoring of influenza-like illness (ILI) in the general population of the Netherlands during the 2003-2004 influenza season. BMC Public Health, 6:242.
6. van Noort SP, Muehlen M, Rebelo de Andrade H, Koppeschaar C, Lima Loureno JM, Gomes MG. (2007). Gripenet: an internet-based system to monitor influenza-like illness uniformly across Europe. Euro Surveill, 12(7): E5-6.

ThermInfo: Collecting and Presenting Thermochemical Properties

Ana L. Teixeira[1*], Rui C. Santos[2], Francisco M. Couto[1]

[1] *LaSIGE, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, Campo Grande, 1749-016, Lisboa, Portugal*
[2] *Centro de Química e Bioquímica, Faculdade de Ciências, Universidade de Lisboa, Campo Grande, 1749-016 Lisboa, Portugal*

Abstract. Due to the large amount of chemical data available, it is essential to organize them and given the numerous problems derived from applying spreadsheets, it motivated us to construct an improved tool geared to contain large amounts of data.

This paper presents *ThermInfo*, a public web tool that integrates a database based upon a relational data specification for describing structural and thermochemical properties of organic compounds. Its user-friendly web interface allows a text-based searching, compounds insertion, and data management.

At present time, *ThermInfo* contains critically evaluated and thermodynamically consistent experimental thermochemical properties values for about 3,000 unique and non redundant compounds. Interface usability results show that it is easy and fast to learn which improves the efficiency when employing the tool.

In the future, we intend to expand the dataset, integrate other classes of compounds databases, use chemoinformatic tools to combine a structure drawn with textual query terms and use prediction methods of thermochemical properties.

ThermInfo is available at http://www.therminfo.com.

Keywords: Chemoinformatics, chemical database, data management, data integration

1 Introduction

In large part, Chemistry is still an empirical science, building its development on an ever increasing amount of data and information. The quantifiable chemo-information keeps growing exponentially due to constantly refined and optimized experimental technologies. According to *Chemical Abstracts Service*, there are currently more than 35 million compounds known, increasing with more than 1 million each year, and more than 600,000 publications per year are related to Chemical information.[1] Thus, it was realized that this flood can only be managed by exploring it in electronic form.

Chemical research work often requires search in many kinds of compounds properties. The problem lies in looking all up in handbooks and papers which is quite unprogressive and time demanding. Thus, it is very important to develop databases to

* To whom correspondence should be addressed (ateixeira@lasige.di.fc.ul.pt)
1 CAS Statistical Summary: [http://www.cas.org/]

manage and search chemical data (such as thermochemical properties) and make them publicly available. Most databases of thermochemical properties are privately owned, expensive, often not so well administrated, and are available through restricted interfaces that are not suitable for the development of statistical and prediction methods [1-3].

The Molecular Energetics Group[2] of Centro de Química e Bioquímica from Faculdade de Ciências, Universidade de Lisboa, has been collecting and critically evaluating experimental thermochemical data of organic compounds from literature. This evaluation is performed through the comparison of values for similar compounds and/or applying additivity methods, and the data is stored in spreadsheets. However, this process triggers several problems, such as:

1. Referential Data Integrity – when it is needed to change or delete some values of a compound property in multiple rows, the same action needs to be repeated several times. Furthermore if some row is forgotten the data will become inconsistent, ambiguous and loses integrity;
2. Data Redundancy – it is directly connected to the previous problem. The data storing is very ineffective due to the same data being entered various times and thereby growing unnecessarily, thus requiring more computer resources (larger size of data and slower access). A good example of this is storing and summarizing the data for the organic compounds characteristics used in *ThermInfo*, since it is a many-many relationship where 28 columns with characteristic names were needed;
3. Data Validity and Non-Uniformity – humans inserting data can be unreliable and there are many ways of entering the same data. The spreadsheets are unable to efficiently identify data errors and different spellings;
4. Limiting Data View – a spreadsheet displays all rows and columns of the table which is bothersome in case of very large datasets, such as *ThermInfo*'s one. The dataset was composed by 3 different sheets, 57 columns and around 3,000 lines. Additionally, there is a lack of detailed sorting and querying abilities. The spreadsheets are designed to analyze data and sort list items, not for long-term storage of raw data;
5. Performance and Capacity – low efficiency with large datasets due to the lack of indexes for accelerating the process of data search, and memory problems due to the load of unnecessary data;
6. Multiuse and Data Entry – data entry forms are not enabled and when the data cannot be conveniently displayed in tabular view, fields for data entry are placed loosely within the spreadsheet. The multiuse requires a lot of discipline, attention, and knowledge from the users which is ineffective and data can be easily corrupted for larger groups of people;
7. Sharing it with the Scientific Community – it is difficult keeping a single file with all the updates and restricting the users from accessing and updating information;

[2] Molecular Energetics Group web page: http://cqb.fc.ul.pt/menergetics/

8. Evolution – the integration of new data, application of prediction methods, and developing other applications that use the dataset is difficult in a spreadsheet format.

All aspects mentioned above motivated us to develop our own system, *ThermInfo*, to collect and present structural information and thermochemical properties, tuned and adapted to meet our specific needs. The goal is to obtain a high quality system that evolves with time, works efficiently according to the purpose, furthermore overcoming the problems of spreadsheets, which guarantees the highest quality, uniqueness and consistency of each entry, plus cost effective to maintain and enhance. Additionally, *ThermInfo* is expected to lower the costs of research and provide an increased number of useful leads, especially when integrated with properties prediction tools.

The rest of this paper is structured as follows: Section 2 describes basic concepts of Chemistry; In Section 3 we present the *ThermInfo* system and its implementation; Section 4 presents the achieved results and discusses them; Finally, in Section 5, we express our main conclusions and directions for future work.

2 Basic Concepts

High-throughput analytical methods for thermochemistry research generate heterogeneous data formats. *ThermInfo* contains experimental data about organic compounds, which can be divided into three categories:

1. *Structural data* – composed by descriptors that specify the molecular structure of the compound [4]:
   - Molecular ID, is a unique and stable identifier for the entity with the format CONNNNN (N = digit);
   - Compound Name, is the name provided for an entity based on the current recommendations of *International Union of Pure and Applied Chemistry* (IUPAC)[3];
   - CAS Registry Number (CASRN), is a unique numerical identifier created and assigned to a chemical substance by *Chemical Abstract Service* (CAS), it does not have any chemical significance and is assigned in sequential order to assure uniqueness. It has the format NNNNNNN-NN-N (1-7 digits, hyphen, 2 digits, hyphen, 1 digit)[4]. The right-most digit is a check digit used to verify the validity and uniqueness of the entire number and it is calculated by taking the last digit times 1, the next digit times 2, the next digit times 3 and so on, adding all these up, and computing the sum modulo 10. For example, the CAS

[3] IUPAC recommendations: http://www.iupac.org/web/ins/2001-043-1-800
[4] CAS Registry Number [http://www.cas.org/expertise/cascontent/registry/regsys.html]

number of methanol is 67-56-1: the checksum 1 is calculated as (6×1 + 5×2 + 7×3 + 6×4) = 61; 61 mod 10 = 1;

- Molecular Formula, identifies each constituent element of a compound by its chemical symbol and indicates the number of atoms of each element in subscript after the chemical symbol. The atoms are in CHXNOS (X = halogen) order. For example, the molecular formula of Benzenemethanol is C_7H_8O, which indicates that it has 7 atoms of carbon, 8 atoms of hydrogen and 1 atom of oxygen;
- Chemical Structure, is a 2D structural diagram of the compound in JPG format;
- Molecular Weight, is the mass of one molecule of that compound, relative to the unified atomic mass unit;
- Physical State, are the distinct forms that different phases of matter take on, and can be: gas, liquid or crystal;
- SMILES (Simplified Molecular Input Line Entry System), is a specification for describing the structure of chemical molecules using short ASCII strings. An important point about this is that there is a difference between upper and lower cases. For example, the cyclohexane has the SMILES C1CCCCC1, while the benzene has the SMILES c1ccccc1 [5];
- USMILES, is a special and unique SMILES among all valid possibilities for a given compound [6];
- Class, Subclass, Family, are hierarchical classifications according with the compound structure;
- Characteristics, are tags according with the functional groups present in the molecule and other characteristics of the compound.

2. *Thermochemical Data* – is related with the energy released or absorbed in chemical reactions, or in physical state transformations [7]:
   - Standard Molar Enthalpy of Formation and the associated error (in $kJ \cdot mol^{-1}$) for:
     - Crystalline Phase;
     - Liquid Phase;
     - Gas Phase;
   - Standard Molar Enthalpy of Phase Change and the associated error (in $kJ \cdot mol^{-1}$);
   - Observations, free text about the enthalpies values.

3. *References Data* – complete references about the compound data, including: author(s), journal/book title, year, volume, and pages.

3 Design Issues and Implementation

To develop *ThermInfo* we are doing multiple interactions of the following steps: system planning; requirements analysis; design; implementation; and maintenance. It is important to note that all these steps are monitored by users, in the first steps to evaluate their own and system needs, and in the last steps, to evaluate the system design and workability [8].

3.1 Functional Requirements

The major features of *ThermInfo* can be divided into two classes, according to its function (represented in Figure 1 [9]):

1. *Search for a Compound*

   ThermInfo has two search methods available to enable both simple and complex queries:

   - *Simple Search*, provides a single text box that allows the users to search for an organic compound in the database, by entering a search term, such as, its name, molecular formula, molecular ID, CASRN, or SMILES;
   - *Structural Search*, provides multiple search fields that allow users to limit the search results to specific compound characteristics.

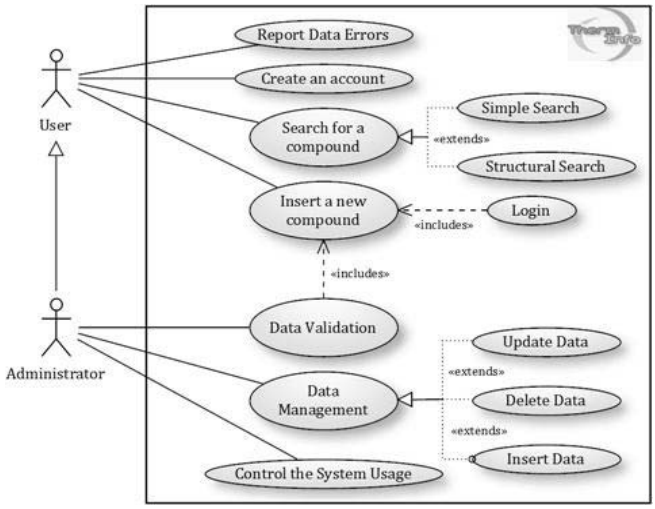


Figure 1 – General overview of the functionality provided by *ThermInfo* in terms of actors, goals and dependencies between use cases.

2. *Data Management*

   2.1 *Insert Data* – allows the users to insert new organic compounds to the database. However, the insertion of the structural, thermochemical, and

reference(s) data is restricted to registered users. Before making this new data public an automatic pre-processing stage and a validation process, by an administrator, are performed in order to filter incorrect, ambiguous or non-unique data. The main purpose of this feature is to support the expansion of the database by the scientific community.

2.2 *Report Data Errors* – if users find errors in *ThermInfo* database, they can report it to an administrator by email.

2.3 *Delete Data* – allows the administrator to search for an outdated compound and delete it from the *ThermInfo* database.

2.4 *Update Data* – allows the administrator to search for outdated/erroneous data about a compound and change it.

2.5 *Validate Data* – allows the administrator to check for the new compounds inserted by *ThermInfo* users and approve or reject the insertion of new data into the *ThermInfo* database.

2.6 *Control Panel* – allows the administrator to monitor the usage and the growth of the database.

3.2 *ThermInfo* Database

We have designed a relational database which has to be flexible by eliminating redundancy, inconsistent dependency and to accommodate heterogeneous data: the structural and thermochemical properties of organic compounds and bibliographic references, selected and carefully evaluated from relevant scientific literature and described in the previous section. Figure 2 shows the Unified Modeling Language (UML) class diagram of the *ThermInfo* database, color coded according to the three categories of data and including the entities, relationships and attributes (with data type) [9].

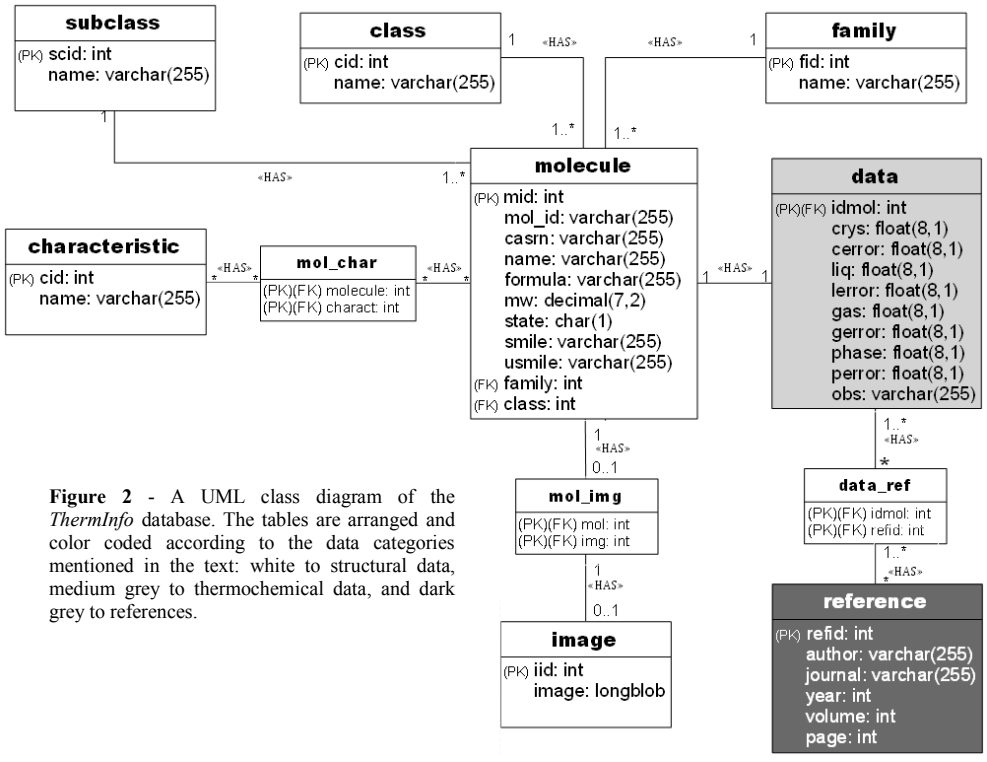


Figure 2 - A UML class diagram of the *ThermInfo* database. The tables are arranged and color coded according to the data categories mentioned in the text: white to structural data, medium grey to thermochemical data, and dark grey to references.

The structural data category is composed by six related entities: the main entity, *molecule*, which contains the data that characterizes the compound and the enforcement of uniqueness assured by the CASRN, since it is a unique descriptor; each molecule can have one *image* with the chemical structure, that is specific to each molecule; each *molecule* can be classified by one *class*, *subclass* and *family*; each *class*, *subclass* and *family* can include several *molecule's;* each molecule can have several *characteristics* or vice versa. The thermochemical properties data category is composed by one entity, *data*, which exists for one molecule (or vice versa). The references category is composed by one entity, *reference*, which can be related to the entity *data*.

3.3 Interface

The web interface was developed according to the requirements described in the previous sections. Similar forms were applied to make it user friendly and fast learning. Figure 3 depicts a composite screenshot from the *ThermInfo* search interface. For searching, the user can use the *Simple Search* form (Figure 3a), typing the search term in the indicated format, select one of the five types of search, and type the security code (that protects our database against malicious scripts). If the search term is not in the correct format or the CASRN does not verify the check digit, the users will receive the adequate message, otherwise they will receive a list of maximum 100 most relevant hits, with the molecular ID, compound name, molecular formula, CASRN, and SMILES (Figure 3b), and a link to the complete information record of the compound (Figure 3c[5]). The *Structural Search* (Figure 4) uses the same procedure, but here the user can specify the structural characteristics of the compound and refine the search.

To *Insert* a new compound, the user needs to create an account or login into an existing one. A form with fields for structural, thermochemical, and references data is available. The data inserted will suffer a pre-processing step and the adequate messages will be presented if something is wrong. The inserted data will be placed in temporary tables waiting for an administrative validation.

To *Delete* or *Update* a compound, the administrator needs to search for the compound using its molecular ID. If it exists, it will display the compound information that can be deleted or changed. The outdated data will be moved to outdated tables.

To *Validate* a compound inserted by users, the administrator is able to see the compound data displayed in a table and mark it to be inserted on the database or to be moved to the outdated tables. The decision is automatically reported to the depositor.

[5] This example can be viewed with more detail at the development website http://xldb.di.fc.ul.pt/biotools/therminfo/compound.php?mid=1511&info=View

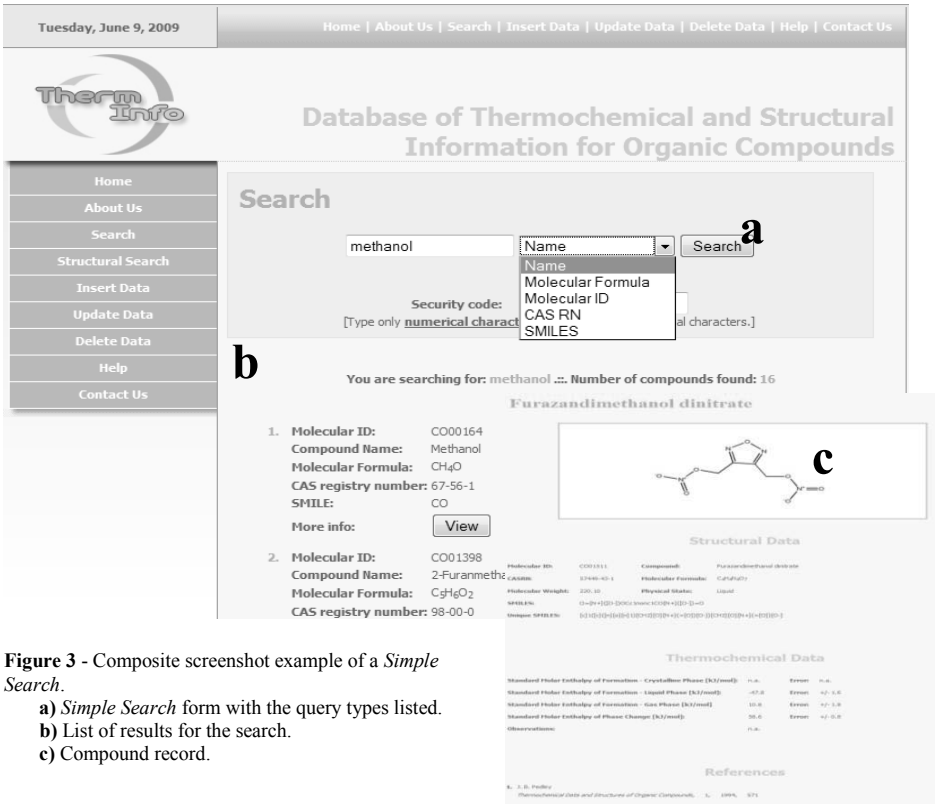


Figure 3 - Composite screenshot example of a *Simple Search*.
a) *Simple Search* form with the query types listed.
b) List of results for the search.
c) Compound record.

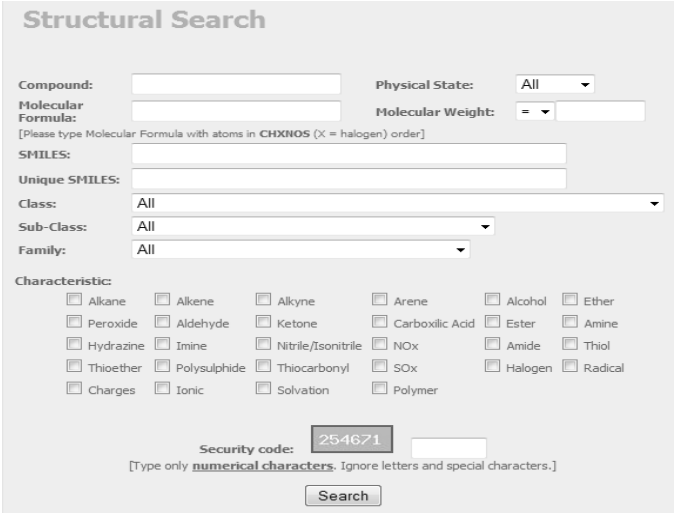


Figure 4 - Screenshot of the *Structural Search* form.

3.4 System Implementation

The *ThermInfo* database was developed using MySQL and phpMyAdmin for systematic and efficient content management and administration over the web. In order to populate the database, we developed Perl scripts to parse and import the data from the spreadsheets.

The user-friendly interface, consisting of dynamic web pages, is developed using HTML, CSS, Java Script, and PHP for data visualization and management. The data is served using Apache web server and the access control is made using *.htaccess* (hypertext access).

ThermInfo is accessible from its official website: http://www.therminfo.com.

4 Results and Discussion

An overview of the *ThermInfo* database statistics in June 2009 is given in Figure 5. It contains around 3,000 unique and non redundant compounds with structural data available for all of them. The histogram examination shows us the completeness and representativeness of the dataset. We expect to see an evolution in number of structures in the next years.

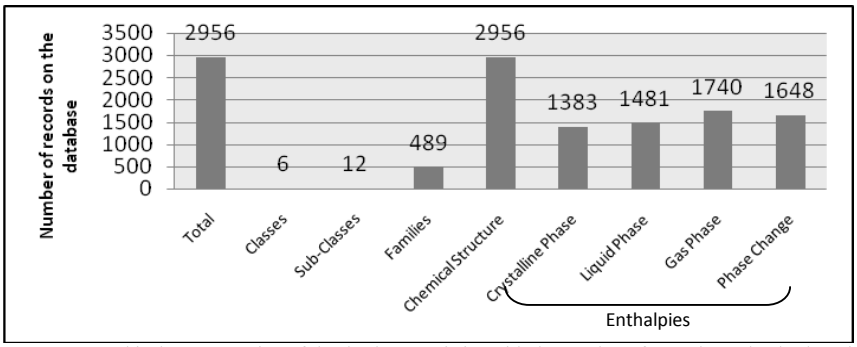


Figure 5 - Graphical representation of the database statistics with the number of records on the database in the different categories of the dataset.

Our implementation is continuously being tested by group members with different backgrounds and is under improvement based on their feedback. An initial low fidelity prototype of the *ThermInfo* interface system was developed, in order to provide directions as initial design choices for layout.

Then, with the high fidelity prototype we did a usability evaluation with 10 participants with high informatics knowledge, and 6 participants with high chemistry knowledge. To evaluate the usability of the interface, it was determined to carry out three tasks based on the features available to users of *ThermInfo*: *Simple Search* (search for a compound using a SMILES string); *Structural Search* (search for a compound based in four structural fields); and *Insert Data* (insert a new compound into de database filling five fields of the form). The evaluation focused on the time the

user takes to perform the task and the number of errors committed. A questionnaire was also done, with the purpose of evaluating the easiness of the learning tasks, memorization of commands, and satisfaction in use. This questionnaire was focused in three opinion scale questions (from zero to five), each one aiming to evaluate the three points mentioned [10-11].

The results of usability evaluation are specified in Table 1. We obtained good times in performing the tasks proposed and few errors for the three tasks. The averages are similar in the two user groups. It is important to verify that for the last task (new compound insertion), although more complex than the first one (simple search), the users took shorter times to perform it and with fewer errors. This means that the similarity between the different forms/commands to realize the tasks allows a fast learning and improves the efficiency. The three subjective opinion questions, about the facility, memorization and satisfaction in use obtained very good results, all values higher than 4. All the users agreed that in future utilizations of *ThermInfo* they would not commit the same errors. All suggestions, comments and errors were taken into consideration for a new phase of interface improving.

Table 1 - Results of usability evaluation tests. The results are presented to all participants, group of information technologies experts (I) and chemistry experts (Chem).

User Group	N	Task†	Time average (seconds)	*St.dev.**	Nr. of errors	*St.dev.**	Facility (0-5)	Memorization (0-5)	Satisfaction (0-5)
All	16	1	66.4	18.2	0.6	0.8	4.5	4.9	4.4
		2	81.2	18.5	0.4	0.5			
		3	64.1	11.9	0.1	0.3			
I	10	1	58.9	15.6	0.3	0.5	4.3	5	4.4
		2	77.0	18.1	0.3	0.5			
		3	59.0	12.1	0.1	0.3			
Chem	6	1	77.7	16.8	1	1.1	4.8	4.8	4.3
		2	88.3	18.3	0.5	0.6			
		3	72.5	4.1	0.2	0.4			

† Task 1 is related with the *Simple Search*, task 2 with *Structural Search*, and task 3 with *Insert Data*.

* St. dev. – standard deviation of the mean value.

5 Conclusions and Future Work

This paper presented *ThermInfo*, a public web tool for collecting and presenting structural, thermochemical and bibliographic data of organic compounds. The tool was implemented in order to store the data into a relational database overcoming the problems presented by the use of spreadsheets. In addition, it provides a user and administrator interface for searching, inserting and managing the data. The development of this tool was a non-trivial task since it required collaboration between informatics and chemists.

We showed that *ThermInfo* will be a useful tool as an easy and free access source point for data fetching, which will be inevitable for chemical research and current

needs. According to user surveys, it provides quick response times, as well as an intuitive and flexible interface. Compared with reputable commercial chemistry databases, *ThermInfo* is still a small database, however, its strength lies in the quality (due to the critical evaluation/validation of the data by thermochemists) and in the accessibility of the data provided.

While we will make significant efforts to screen the experimental data added, we request that the community participates in this process, expanding the thermochemical dataset. The organic compounds database compilation will be followed by organometallic, radicals, and inorganic databases. Furthermore, the integration with existing open source chemoinformatics tools, such as, JChemPaint[6] (to draw chemical structures and combine it with textual query terms to further restrict searches), and OpenBabel[7] (to convert a SMILES string in other chemical structure formats) will expand the capabilities of *ThermInfo*. As future work, we also expect the implementation and integration of prediction methods for thermochemical properties based on structure-energetics relationships [12-14].

Acknowledgments. This work was supported by FCT, through the Multiannual Funding Program. R.C.S. gratefully acknowledges Fundação para a Ciência e a Tecnologia (FCT) for a post-doctoral grant (SFRH/BPD/26610/2006).

References

1. Chen, W. L.: Chemoinformatics: Past, Present, and Future. J. Chem. Inf. Model., 46, 2230-2255 (2006).
2. Engel, T.: Basic Overview of Chemoinformatics. J. Chem. Inf. Model., 46, 2267-22774 (2006).
3. P.J. Linstrom, W.G. Mallard (Eds.), NIST Chemistry WebBook, NIST Standard Reference Database Number 69, National Institute of Standards and Technology, Gaithersburg MD, <http://webbook.nist.gov> (accessed in May 2009).
4. Degtyarenko, K., de Matos, P., Ennis, M., Hastings, J., Zbinden, M., McNaught, A., Alcántara, R., Darsow, M., Guedj, M. and Ashburner, M.: ChEBI: A Database and Ontology for Chemical Entities of Biological Interest. Nucleic Acids Res., D344–350 (2008).
5. Weininger, D.; SMILES, A Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. J. Chem. Inf. and Comp. Sciences, 28, 31-36 (1988).
6. Weininger, D., Weininger, A., Weininger, J. L.: SMILES 2. Algorithm for Generation of Unique SMILES Notation. J. Chem. Inf. and Comp. Sciences, 29, 97-101 (1989).
7. a) Pedley, J. B., Naylor, R. D., and Kirby, S. P.: Thermochemical Data of Organic Compounds. 2nd ed., Chapman and Hall, London (1986). b) Pedley, J. B.:

[6] JChemPaint: http://apps.sourceforge.net/mediawiki/cdk/index.php?title=JChemPaint
[7] OpenBabel: http://openbabel.org/wiki/

Thermochemical Data and Structures of Organic Compounds. TRC Data Series, vol. 1, College Station, TX, (1994).
8. Modha, J., Gwinnett, A., Bruce, M.: A Review of Information Systems Development Methodology (ISDM) Selection Techniques. Omega, 18, 473-490, (1990).
9. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison Wesley Longman, Reading, MA, 1999.
10. Dix, A., Finlay, J., Abowd, G. D., Beale, R.: Human Computer Interaction. 3rd ed., Prentice Hall, 2003.
11. Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T.: Human Computer Interaction. Addison Wesley, 1994.
12. Poling, B. E., Prausnitz, J. M., O'Connell, J. P.: The Properties of Gases and Liquids. 5th ed., McGraw-Hill: Singapore, 2001.
13 Leal, J. P.: Additive Methods for Prediction of Thermochemical Properties. The Laidler Method Revisited. 1. Hydrocarbons. J. Phys. Chem. Ref. Data, 35, 55-76 (2006).
14. Santos, R. C., Leal, J. P., Martinho Simões, J. A.: Additivity Methods for Prediction of Thermochemical Properties. The Laidler Method Revisited. 2. Hydrocarbons Including Substituted Cyclic Compounds. J. Chem. Thermodyn., (2009), doi:10.1016/j.jct.2009.06.013.

A Comparison of Different Approaches for Assigning Geographic Scopes to Documents

Ivo Anastácio, Bruno Martins, and Pável Calado

Instituto Superior Técnico, INESC-ID,
Av. Professor Cavaco Silva, 2744-016 Porto Salvo, Portugal
{ivo.anastacio,bruno.g.martins,pavel.calado}@ist.utl.pt

Abstract. In this paper, we compare different methods for the automatic assignment of geographic scopes to Web pages, based on placenames mentioned in the text. The methods under study are the Yahoo! Placemaker Web service, the hierarchy-based method originally proposed for the Web-a-Where system, the spatial overlap-based method originally proposed in the GIPSY project, the graph-based method originally proposed in the GREASE project, and three simple baseline methods corresponding to using the most frequently occurring place, the spatial area that covers all mentioned places, or the spatial area that covers all mentioned non-outlier places. The task under study may be included into the broader problem of Geographic Information Retrieval. Experiments were carried out on Web pages from the Regional Section of the Open Directory Project, comparing the automatically assigned scopes against those assigned by human editors. The results show that the Web-a-Where method gives the best results, closely followed by the GraphRank method and by the baseline based on to the most frequent occurring place.

Sumário Este artigo compara diferentes métodos para atribuir automaticamente âmbitos geográficos a páginas Web, com base nos nomes de locais mencionados no texto. Os métodos avaliados são o do serviço Web Yahoo! Placemaker, o método proposto para o sistema Web-a-Where baseado em hierarquias, o método proposto no projecto GIPSY baseado na sobreposição espacial, o método proposto no projecto GREASE com base em grafos, e três *baselines* correspondentes a atribuir o lugar mais frequentemente mencionado, a região que cobre todos os locais mencionados, e a região que cobre todos os locais mencionados que não são *outliers*. A tarefa de atribuir âmbitos geográficos pode ser incluída no problema mais abrangente da Recuperação de Informação Geográfica. As experiências utilizaram as páginas Web presentes na secção Regional do Open Directory Project, comparando-se os âmbitos atribuídos automaticamente com os atribuídos por humanos. Os resultados mostram que o método Web-a-Where produz os melhores resultados, seguido de perto pelo GraphRank e pela *baseline* com base no local mais frequente.

Key words: Cross-Method Comparison, Geographic Text Mining

This work was partially supported by the FCT (Portugal), through project grant PTDC/EIA/73614/2006 (GREASE-II).

1 Introduction

Recently, Geographical Information Retrieval (GIR) has captured the attention of many researchers that work in fields related to text mining and data retrieval. Most text documents can be said to be related to some form of geographic context [9]. However, exploring this information presents non-trivial problems, due to the inherent ambiguity of natural language (e.g., placenames often have other non geographic meanings, different places are often referred to by the same name, and the same places are often referred to by different names). Handling place references in text has nonetheless been addressed by several previous works, with the aim of supporting subsequent GIR processing tasks.

The automatic assignment of geographic scopes to Web documents, based on the place references that are present in the text, is an example of a complex GIR problem that has been getting increasing attention. Given a set of diverse geographic regions, corresponding to the placenames mentioned in a given Web page, the problem concerns finding the geographic region that best summarizes and describes them all. The use of the word *diverse* proposes that the set of regions referenced in a document may not be trivial to combine, since there may be some errors in the disambiguation of the placenames mentioned in the text, and not all of the regions resulting from the disambiguation may correspond to sub-areas of a single well-bounded place, like a city. However, it is assumed that the set is somewhat coherent—we do not expect that some place references correspond to regions in Lisbon, Portugal while other place references in the same document correspond to regions in Sidney, Australia. While several different strategies have been proposed in the past, there is nowadays no clear information about the trade-offs involved in choosing a particular algorithm. Each different algorithm makes specific assumptions, therefore resulting in different approximations for the geographic scope of the documents.

This paper presents an empirical comparison of different methods for the automatic assignment of geographic scopes to Web pages, based on placenames mentioned in the text. The GeoCLEF campaign has addressed the black-box evaluation of GIR systems, some of them considering components for scope assignment [4]. However, to the best of our knowledge, no cross-method comparison on the specific problem of scope assignment has ever been reported. The methods considered in this study are the Yahoo! Placemaker Web service, the hierarchy-based method originally proposed for the Web-a-Where system, the spatial overlap-based method originally proposed in the GIPSY project, the graph ranking method originally proposed in the GREASE project, and three simple baseline methods corresponding to using the most frequently occurring place, the spatial area that covers all mentioned places, or the spatial area that covers all mentioned non-outlier places. The cross-method comparison was carried out by comparing the automatically assigned scopes against those assigned by humans. As reference dataset, we used a collection of 6,000 Web pages extracted from the Regional Section of the Open Directory Project[1], which contains

[1] `http://dmoz.org/`

documents classified according to both broad (e.g., United States) and narrow administrative regions (e.g., Plymouth, a city in Minnesota).

The rest of this paper is organized as follows: Section 2 discusses the recognition and disambiguation of place references in text, a necessary pre-processing step in the task of scope assignment. Section 3 details the different scope assignment approaches that are the target of this study. Section 4 presents the experimental evaluation. Finally, Section 5 presents our conclusions, also giving some ideas for future evaluation studies.

2 Recognizing Place References in Text

One of the particular problems that has been extensively explored in the area of GIR relates to handling place references, a.k.a. *geotagging text*. This is a crucial task for other GIR-related problems, such as the determination of the geographic scope of Web pages. Handling place references requires recognizing the mentions to places given over text, by delimiting their occurrences, as well as disambiguating these occurrences into the corresponding locations on the surface of the Earth, by assigning geospatial coordinates to the place references. The main challenges involved in both sub-tasks are related to ambiguity in natural language. Amitay et al. characterized ambiguity problems according to two types, namely *geo/non-geo* and *geo/geo* [3]. Geo/non-geo ambiguity refers to the case of placenames having other non-geographic meanings, since some of very common words are also place names (e.g., Turkey the country, or Reading in England). Geo/geo ambiguity arises when two distinct places have the same name. For instance almost every major city in the Europe has a sister city of the same name in the (so-called) New World.

Leidner surveyed a variety of approaches for handling place references on textual documents [2]. Most methods usually rely on finding the placename in a dictionary of known locations (a *gazetteer*), together with natural language processing heuristics such as default senses (i.e., disambiguation should be made to the most important referent, estimated based on population counts) or geographic heuristics such as spatial minimality (i.e., disambiguation should minimize the bounding polygon that contains all candidate referents). Place reference resolution technology is nowadays mature, and commercial services offering this type of functionalities are starting to appear.

Metacarta[2] is an example of a commercial company that sells state-of-the-art geographic information retrieval technology. The company also provides a freely-available Web service that can be used to recognize and disambiguate place references over text. An early version of the Metacarta geotagger has been described by Rauch et al. [5]. The Yahoo! Placemaker Web service also provides a functionality for geotagging text. The Yahoo! Placemaker is used in all our experiments to handle the recognition and disambiguation of place references, and will be further described in Section 3.1.

[2] `http://metacarta.com/`

3 Scope Assignment Approaches

This section presents the different scope assignment methods that are the focus of this study, also discussing particular issues regarding the implementations that were used in the experiments.

3.1 The Yahoo! Placemaker Web Service

Yahoo! Placemaker[3] is a geotagging web service that provides third-party developers the means to enrich their applications or Web sites with geographic information. The service is able to identify, disambiguate, and extract place names from unstructured and semi-structured documents. It is also capable of using the place references in a document, together with a pre-determined set of rules, to discover the geographic scope that best encompasses its contents. Thus, given a textual document, Yahoo! Placemaker returns unique *Where-on-Earth* identifiers (WOEIDs) for each of the named places and scopes. Through these identifiers, one can use the Yahoo! GeoPlanet[4] Web service to access hierarchical information (i.e., containing regions) or spatial information (i.e. centroids and bounding boxes).

There are two flavors of document scopes in Placemaker, namely the geographic scope and the administrative scope. The geographic scope is the place that best describes the document. The administrative scope is also the place that best describes the document, but is of an administrative type (i.e., Continent, Country, State, County, Local Administrative Area, Town, or Suburb). Since the reference document collection that we used for our experiments only contains documents assigned to administrative regions, we limited our cross-method comparison to using Placemaker's administrative scopes.

Placemaker is a commercial product and not many details are available regarding its functioning. However, some information about the service is available in the Web site, together with its documentation. For instance, the Web site claims that when the service encounters a structured address, it will not perform street level geocoding but will instead disambiguate the reference to the smallest bounding named place known, frequently a postal code or neighborhood. The Web site also claims that besides place names, the service also understands geography-rich tags, such as the W3C Basic Geo Vocabulary and HTML microformats[5]. However, no details about the rules that are used in the scope assignment process are given in the documentation for the service.

The Placemaker Web service accepts plain text as input, returning a XML document with the results. The service has an input parameter that allows users to provide the title of the document separately from the rest of the textual contents, weighting the title text as more representative. In our experiments, we used the Web service as a black-box to assign scopes to the Web documents, using the option that weights the title text as more important than the rest.

[3] `http://developer.yahoo.com/geo/placemaker/`
[4] `http://developer.yahoo.com/geo/geoplanet/`
[5] See `http://microformats.org/wiki/geo` or `http://microformats.org/wiki/adr`

3.2 The GIPSY Scope Assignment Method

In one of the pioneering works in the area of GIR, Woodruff and Plaunt proposed a technique for computing the geographic scope of a given textual document based on the place references discovered in the text [2]. Their method is based on disambiguating the place references into their respective bounding polygons. The geographic scope of the document is afterwards computed using the overlapping area for all the polygons, trying to find the most specific place that is related to all the place references made in the text.

Consider, for instance, the following example. A given document contains references to Portugal, to the city of Lisbon, and to the Iberian Peninsula. After disambiguation, each of these place references is represented as a bounding box. For the GIPSY algorithm, the bounding boxes are seen as thick polygons, with a base positioned in an (x, y) plane, but extending upwards a distance of z, to a higher parallel plane. One by one, the three bounding boxes corresponding to the place references are analyzed by the GIPSY algorithm, in order to build a skyline of bounding boxes. Three different cases can occur:

- When adding the bounding box for Portugal, it would not intersect with other bounding boxes. Portugal would simply be laid at $z = 0$;
- When adding the bounding box for Lisbon, it would be completely contained within a bounding box which already exists on the skyline, in this case the bounding box for Portugal. Lisbon would be laid on top of the Portuguese bounding box, i.e. its base would be positioned at a higher z plane.
- When adding the bounding box for the Iberian Peninsula, it would intersect other bounding boxes but it would not be wholly contained. The bounding box would first be split into multiple polygons. Then, the intersecting polygons would be laid on top of the existing bounding boxes, one on top of Lisbon and another on top of Portugal, and the non-intersecting polygon (continental Spain) would be laid at a lower level.

Finally, all the bounding boxes would be sorted according to their z order and the highest ranking bounding box is selected as the scope. In our example, the resulting scope would correspond to the area of Lisbon.

The Yahoo! Placemaker Web service was used to recognize place references in the documents and disambiguating them into bounding boxes. The Java Topology Suite [8] provided the required functionality related to spatial computations.

3.3 The Web-a-Where Scope Assignment Method

In the context of the Web-a-Where project, Amitay et al. proposed a technique for assigning Web documents to the corresponding geographic scopes [3]. Their technique leverages on part-of relations among the recognized place references, provided by a hierarchical gazetteer. The basic idea is that, for instance, if several cities from the same country are mentioned, this might mean that this country is the scope, i.e. the algorithm tries to generalize from the disambiguated place references. More specific places are scored higher if they are the only places

mentioned, but at the same time we also permit a general region to be chosen if several different places in it are mentioned, with no specific emphasis on any.

The algorithm starts by placing the recognized place references in a locational hierarchy. By looping over the disambiguated references, the algorithm aggregates the importance of the various levels in the hierarchy. The levels are then sorted by importance and the highest ranked level is returned as the scope.

Consider, for instance, the following example. After place reference disambiguation, a document contains a reference to the city of Lisbon (i.e., `Europe/Portugal/Lisbon`) with confidence 0.5, and a reference to Portugal (i.e., `Europe/Portugal`) with confidence 0.8. A taxonomy is first built from the disambiguated place references. Then, the taxonomy nodes are weighted according to the disambiguated place references, where `Europe/Portugal/Lisbon` gets a weight of 0.5^2 and `Europe/Portugal` gets a weight of 0.8^2. This quadratic scoring function increases the relative weight of very confident disambiguations.

After assigning the initial weights we propagate scores to the parent regions in the taxonomy, by adding the scores of their respective sub-regions. Thus, we add $0.5^2 \times 0.7$ to `Europe/Portugal` and $0.5^2 \times 0.7^2 + 0.8^2 \times 0.7$ to `Europe`. The multiplying discount parameters correspond to those originally reported in the Web-a-Where paper. Finally, we select the highest scoring taxonomy node as the scope to assign (Portugal, in our example).

We used the Yahoo! Placemaker Web service as the means for recognizing place references in the documents and disambiguating them into nodes in the hierarchical gazetteer used in the Yahoo! GeoPlanet platform.

3.4 The GraphRank Scope Assignment Method

In the context of the GREASE project, Martins and Silva proposed a scope assignment method based on a graph-ranking approach [6]. The idea was to represent the gazetteer used for place reference disambiguation as a graph, where the nodes correspond to different places and the edges correspond to semantic relationships (*part-of*, *containment* or *adjacency*) between places. Nodes on this graph can be weighted according to the occurrence frequency of place references in a document, and edges can be weighted according to the relative importance of the different types of relationships. A graph-ranking algorithm, *PageRank*, is then applied to this graph, and finally the highest ranked node is selected as the scope. In case of ties, the node connected to the highest number of edges is selected. By propagating scores across the graph, this algorithm tries, at the same time, to generalize and to specify from the available information, in order to find the region that best reflects the scope of the document. For computing the PageRank score, we used the open-source weighted PageRank implementation made available by the Laboratory for Web Algorithmics of the University of Milan [7]. Since this implementation does not allow for weighted nodes, we instead use self-edges, one for each occurrence of a given place in the document.

Consider the following example. A given document contains references to United States and to Los Angeles, which are extracted and disambiguated with confidences of 0.9 and 0.8, respectively. In order to generate the graph, we would

first find the hierarchical parents of the references that are made in the document, the neighboring places to the document references, and the hierarchical parents for these neighboring places. The places discovered through the above procedure would be the nodes of the graph and the relationships between them would be used to produce directed edges between the nodes. For all the nodes with no outlinks (i.e., the roots and the leaf nodes) we would add artificial edges to all other nodes in the graph. The part-of, containment, and adjacency edges would all get a value of 0.4, and artificial edges 0.01. These weights were tunned empirically, and a challenge for future work consists of using automated approaches for tunning these parameters. United States and Los Angeles would also have edges to themselves, with weights equal to their confidence scores. The PageRank algorithm would then be applied and, in the end, the highest scoring node would be selected as the scope.

The Yahoo! Placemaker Web service was used for recognizing place references in the documents and disambiguating them into nodes in a hierarchical gazetteer. The complementing Yahoo! GeoPlanet Web service was used to retrieve the parent and neighboring regions for each of the place references that are made in the document. The graph is built by considering all this information.

3.5 Baseline Scope Assignment Methods

The previously described methods make non-trivial assumptions about how place references should be combined to discover the geographic scope of a document. In order to assess what are the gains introduced by these assumptions, we implemented three simple baseline methods, which we now describe:

- **Assigning the scope according to the most frequently occurring place reference** - The number of times a place is referenced in a document reflects the importance of that place to the document's subject. We therefore experimented with a simple scope assignment method that chooses the most frequently occurring place reference as the scope. In case of ties, the place reference corresponding to the largest area is chosen.
- **Assigning the scope according to the bounding box that covers all the place references** - The different place references made in the document should all contribute to the document's scope. We therefore experimented with a simple scope assignment method that computes the bounding box that covers all the place references made in the document.
- **Assigning the scope according to the bounding box that covers all the place references that are not outliers** - This is a refinement of the previous strategy, in the sense that not all place references should contribute to the scope, but only the place references that are somewhat interrelated. The idea is to be able to filter the errors made while recognizing and disambiguating place references, as well as filtering out the place references that are only tangential to the content of the document. We first compute the average centroid point for all the place references made in the document, as well as the average distance between the place references and this centroid.

	All Pages	Countries	States	Cities
Number of documents	6000	2000	2000	2000
Number of ODP sub-classes	1440	692	665	303
Number of different regions	1127	1	51	1075
Average document length (bytes)	4143	4235	3841	4354
Average number of place references	9.2	9.1	9.1	9.4

Table 1. Statistical characterization for the test collection of ODP documents.

Then, we filter out those place references whose centroid is at a distance that is greater than twice the average distance value. Finally, we assign a scope corresponding to the bounding box that covers all the remaining place references, if none the closest is chosen. This baseline is inspired on a technique proposed by Smith and Crane for placename disambiguation [10].

We implemented the above three strategies by using the Yahoo! Placemaker Web service to recognize place references in the documents and disambiguate them into bounding rectangles. The Java Topology Suite [8] provided the required functionality related to spatial computations.

4 Comparative Experiments

In this section, we describe the details of our empirical evaluation. This includes the experimental design, the datasets, and the evaluation metrics that were considered, as well as the results of the experiments that evaluate the effectiveness of the methods under study.

4.1 Dataset

We evaluated the algorithms described in Section 3 for assigning documents to geographic scopes by comparing their assignments to those of the human editors of the Open Directory Project (ODP). Specifically, we selected a random sample of 6,000 Web-pages from the ODP's `Regional/North_America/United_States` section, that were written in English, were larger than 2 KBytes, and contained at least one geographic reference. The collection contains documents classified according to both broad (e.g., United States) and narrow (e.g., Plymouth, Minnesota) administrative regions. Table 1 presents a statistical characterization of the test collection, separating the pages according to the type of geographic scope to which they belong (i.e., country, state, and city).

4.2 Evaluation Metrics

We propose to compare the different approaches by measuring the distance and the relative overlap between the geographic scope that was assigned by the algorithms and the geographic scope that was assigned by the human editors of

	Avg. Distance	Std.dev. Distance	Avg. Overlap	Std.dev. Overlap	Accuracy (D=0 Km)	Accuracy (D<100 Km)	Accuracy (O>0.75)
Placemaker Admin.	1030	**1460**	0.42	0.49	0.37	0.45	0.39
Web-a-Where	**955**	1890	0.48	0.49	**0.47**	0.54	0.47
GIPSY	1265	2247	0.25	0.41	0.14	0.4	0.19
GraphRank	1083	1955	0.48	0.49	**0.47**	0.53	**0.48**
Covering Area	2655	3009	0.25	**0.38**	0	0.21	0.18
Most Frequent	1093	2331	**0.49**	0.49	0.37	**0.55**	0.43
Non-outliers	1740	2826	0.36	0.46	0.24	0.39	0.34

Table 2. Comparison of human-assigned versus automatically assigned scopes.

the Open Directory Project. Besides the average distances and overlaps, we also compute their standard deviation. Thresholding the distance results at different levels, we also measure results according to the standard information retrieval metric of accuracy (i.e., the proportion of correct results given by the algorithm).

4.3 Comparison Results

Figure 1 and Table 2 show the obtained results for the cross-method comparison. Figure 1 plots, for each algorithm, the distances from the assigned scopes to the real scopes of the documents. Each value on the x axis corresponds to a document in the collection, and documents are sorted according to the distance. Table 2 summarizes the values obtained using all the evaluation measures.

The charts show that the covering area baseline produces more errors. In most of the considered approaches, more than half of the documents are assigned to a nearby scope. The GIPSY method and the most frequent baseline produce more errors on pages whose scope corresponds to countries, whereas the other methods have errors equally distributed across countries, states, and cities.

The Web-a-Where method produced the best overall results. The average distance and accuracy to the correct scope were 955 Km and 47%, respectively. The baseline considering the most frequent place reference provided very competitive results, outperforming all other approaches when assigning scopes with an error below 100 Km. The baseline with the most frequent place also obtained the best average overlap with the correct scope (0.49). The GraphRank method did well, matching Web-a-Where's accuracy for exact matches and obtaining the best accuracy for approximate overlaps.

Since the methods with best performances exhibit different types of problems (e.g., the most frequent baseline fails on countries), their combination seems like a promising approach. For instance, Web-a-Where tends to fail when incorrectly generalizing to a broader area, while the most frequent baseline does not generalize when it should.

We also analyzed how accuracy varies according to the number of place references contained in the documents. Figure 2 presents the obtained results, showing that the errors increase with the number of different place references contained in the document. The baseline method corresponding to the covering area is particularly sensitive to this parameter.

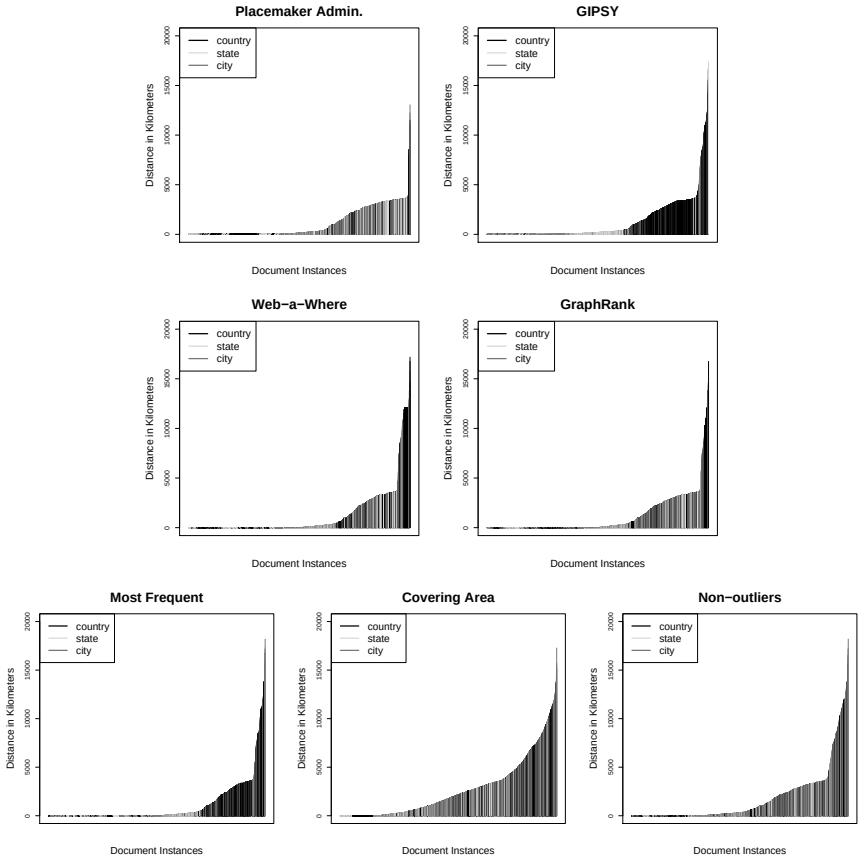


Fig. 1. Distances between the human-edited and automatically assigned scopes.

When interpreting the results, it should be noted that the scope assignment algorithms use the information provided by the Yahoo! geotagger about the individual places mentioned in the text. Thus, any errors made by the geotagger influence the outcome of the scope assignment methods (i.e., this test only evaluates geotagging plus scope assignment as a whole). Nevertheless, this does not invalidate the goal of the experiments, which is to compare the scope assignment methods under the same conditions.

5 Conclusions and Future Work

In this paper, we compared different methods for the automatic assignment of geographic scopes to Web pages, based on placenames mentioned in the text.

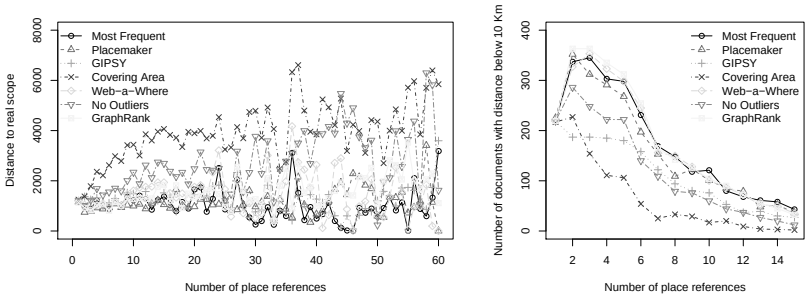


Fig. 2. Correlation between number of place references and scope accuracy.

This is an important pre-processing stage for geographic IR applications. Experimental results showed that overall the Web-a-Where method gives the best values. Nonetheless, this method is closely followed by a simple baseline that assigns the most frequent place reference as the geographic scope, as well as by the graph-based approach.

To the best of our knowledge, this was the first survey and cross-method comparison made in this particular domain. While our results are interesting, there are also many open questions. Bellow, we list what we consider to be the most interesting paths for future work.

- Optimizing the parameters used in some of the considered approaches. For instance, Web-a-Where uses a hierarchical discount parameter and GraphRank uses different weights for each of the geographic relationship types. Ideally, these parameters should be optimized according to a principled approach (e.g., through simulated annealing, genetic programming, or other optimization methods [11]).
- Devising particular tests to see if some of the algorithms are better for some types of documents. In our experiments, we have seen that all algorithms are similarly affected by the number of place references made in the documents. However, it remains to be seen if the algorithms are all equally robust to the ambiguity problems that may arise in different types of documents. If indeed it is the case that some algorithms are better then others in particular cases, then perhaps we can explore machine learning approaches to select the most appropriate scope assignment method to use in each case. Taking the best algorithm for each document in the test collection, we would get an accuracy of 70%, showing that this is indeed a promising alternative.
- Devising strategies for combining the results produced by the different scope assignment methods into a single scope. Our experiments showed significant differences in the algorithms and it would be interesting to see if a combination of the best algorithms could lead to better results. Possible combination

strategies include taking the bounding box that covers all scopes, or taking the bounding box from their overlapping area.
- Devising experiments to measure the trade-offs in selecting more than one scope for each document. Many documents can not be naturally summarized into a single scope, and some applications may also benefit from having multiple scopes. When considering multiple scopes, there is a chance that we are increasing recall. Precision would nonetheless be impacted, since we would select more scopes incorrectly. A particular challenge is finding the appropriate thresholds for selecting candidates as scopes.
- Devising experiments for measuring how the quality of place reference disambiguation influences the quality of the scope assignments. Instead of always relying on a single approach for place reference disambiguation, it would be interesting to experiment with other geotagging approaches, measuring their performance and their impact on the scope recognition performance. It would also be interesting to see if the scope assignment methods are robust enough to work with all the possible place referents, therefore dispensing the pre-processing step of place reference disambiguation.

References

1. Woodruff, A. G., and Plaunt, C. (1994) GIPSY: automated geographic indexing of text documents. Journal American Society Information Sciences, 45(9).
2. Leidner, J. L. (2007). Toponym Resolution: a Comparison and Taxonomy of Heuristics and Methods. PhD Thesis, University of Edinburgh.
3. Amitay, E., Har'El, N., Sivan, R., and Soffer, A. (2004) Web-a-Where: geotagging web content. In Proceedings of the 27th Annual international ACM SIGIR Conference on Research and Development in information Retrieval.
4. Mandl, T., Gey, F., Di Nunzio, G., Ferro, N., Sanderson, M., Santos, D. and Womser-Hacker, C. (2008) An evaluation resource for geographic information retrieval. In Proceedings of the 6th Language Resources and Evaluation Conference.
5. Rauch, E., Bukatin, M., and Baker, K. (2003) A confidence-based framework for disambiguating geographic terms. In Proceedings of the HLT-NAACL 2003 Workshop on Analysis of Geographic References.
6. Martins, B., and Silva, M. J. (2005) A Graph-Ranking Algorithm for Geo-Referencing Documents, In Proceedings of the 5th IEEE International Conference on Data Mining.
7. Boldi, P., Santini, M., and Vigna, S. (2005) PageRank as a function of the damping factor. In Proceedings of the 14th International World Wide Web Conference.
8. Johansson, M., and Harrie, L. (2002) Using Java Topology Suite for real-time data generalization and integration. In Proceedings of the 2002 workshop of the International Society for Photogrammetry and Remote Sensing.
9. Jones, R., Zhang, W. V., Rey, B., Jhala P., and Stipp E. (2008) Geographic intention and modification in Web search. International Journal of Geographical Information Science, 22(3).
10. Smith, D. A. and Crane, G. (2001) Disambiguating Geographic Names in a Historical Digital Library. In Proceedings of the 5th European Conference on Research and Advanced Technology For Digital Libraries.
11. Spall, J. C. (2003) Introduction to Stochastic Search and Optimization, Wiley-Interscience.

Sessão 3A: XML: Aplicações e Tecnologias Associadas (XATA)

Schema Languages for XML

Hugo Manguinhas[1],

[1] INESC-ID – Instituto de Engenharia de Sistemas e Computadores,
Apartado 13069, 1000-029 Lisboa, Portugal
hugo.manguinhas@ist.utl.pt

Abstract. This paper addresses the problem of the characterization of XML schema languages. The motivation is to offer enough information for one to choose the schema language that best fits the schema constraint requirements. XML is a language developed to encode data in a structured human readable way. Its simple syntax allows the representation of any sort of information from any particular domain of interest. In order to be portable across platforms and systems, XML documents often need to be associated to a well defined structure declaration: a schema. A schema is therefore a formal way to declare the syntax and to define the relationships and semantics of the attributes of an XML document, and for that purpose XML schemas languages can be defined. This paper presents an overview of the actual state of the art in schemas languages and a comparative analysis of them. As a result of this work valuable conclusions were reached to anyone willing to formalize schemas for XML documents.

Keywords: Structured Information, Document Engineering, XML, Schema, Schema Language, Grammar, Rules, Constraints, DTD, W3C XML Schema, RELAX NG, Schematron.

1 Introduction

This paper addresses the problem of the representation of schemas using XML schema languages. The paper presents an overview and a description of the actual state of the art in schema languages, including a comparative analysis of these languages.

The results of this work can be valuable to anyone willing to express a schema using a XML schema language. The conclusions of this paper might help choosing the schema language and strategies that best fits the schema constraint requirements. XML documents are structured human readable text representations of data. Since they are structured, XML documents are efficient solutions to store and transmit information. But for that purpose, these documents need a well defined structure in order to be portable across platforms and development systems.

One way to accomplish this wellness is by developing **schemas**. The purpose of a schema is to make it possible to describe a class of XML documents using constraints to define the usage and relationship of their underlying foundations, such as elements, attributes, attribute values and text. These constraints come from the rules governing

the domain of interest that the particular XML document is supposed to represent. From an object oriented perspective, schemas can be seen as classes (template for the objects) and XML documents as objects, instantiations of those classes. Essentially, schemas can be used to catalogue classes of XML documents. Like in common object paradigms, schemas can also be composed of other schemas to achieve higher levels of abstraction.

Schema **languages** have been developed to better represent the rules and constraints contained in schemas. For that, schema languages rely on **grammars** to define, in a logical form, all the constraints contained in a schema. As a consequence, those definitions can be used to **validate** documents that are instances of these schemas. Many attempts were made to build the ultimate schema language able to make it possible to express all the required constraints of any schema. This paper will focus a greater attention in four of these languages: Document Type Definition (here mentioned in short as DTD), W3C XML Schema (here mentioned in short as WXS), RELAX NG and Schematron. These four languages were chosen because they accurately represent the approaches that have guided the evolution of schema languages over the years.

The next section presents a short description of the four schema languages that were subject of analysis (Section 2). After that, the concept of grammar and its relevance for schemas is then presented (Section 3), followed by a description of their schema constraints according to their lexicon, syntax and semantics (Section 4). Finally, the paper compares the main characteristics of these languages (Section 5), presents other complementary work (Section 6), and finishes with conclusions and future work (Section 7).

2 Schema Languages

In this section, the four languages, which will be the focus of this paper, are introduced.

XML Document Type Definition [1] (DTD in short) is a subset of the original SGML DTD [2] (the schema mechanism for SGML). It was developed along with the XML specification to be the *de facto* standard for XML schema languages. It was the first step towards the development of a schema language as we know it. XML DTD is a W3C Recommendation since February 1998 and its current version is 2.10[1]. The DTD document declaration can appear internally or externally to the XML document. The DTD schema language defines the XML document structure through a list of element and attribute declarations, entities, references, comments and notations. The last four structure entities are important to the XML document encoding and processing but are irrelevant for the schema definition.

W3C XML Schema (here WXS in short) [3][4][5], sometimes informally called XML Schema Definition (XSD), which is the name given to a WXS instance document. It was the first wide-spread attempt to replace DTDs with a new schema language that would fit the increasing requirements. WXS is a W3C recommendation

[1] http://www.w3.org/2002/xmlspec/dtd/2.10/xmlspec.dtd.

since May 2001 and was the first separate schema language for XML to achieve Recommendation status by the W3C. WXS is similar to the DTD schema language by also defining the document structure through a list of declarations of elements and attributes. Nevertheless, it adds the ability to define types or frames in an object oriented approach that can be applied throughout the source document.

RELAX NG [6] (REgular LAnguage for XML Next Generation) consists of fusion of two earlier languages; RELAX[2] (REgular Language description for XML) designed by Murata Makoto and approved by ISO/IEC Technical Report 22250-1; and TREX[3] (Tree Regular Expressions for XML) designed by James Clark which is a subset of yet another language called XDuce[4]. RELAX NG was developed to be easier to learn and use. Two relevant aspects contributed significantly to these requirements and distinguish it from the other schema languages. The first consists on the language ability to follow the XML document tree-like structure, opposed to its leveraging into a list of structure declarations. The second consists on its ability to uniformly reapply the same primitives to all of the document structures.

The **Schematron Assertion Language** [7] was developed by Rick Jelliffe at the Academia Sinica Computing Centre (ASCC). Schematron is available as Final Committee Draft for the ISO International Standard Organization (ISO/IEC FDIS 19757-3) since October 2004, as part of a broader specification named Document Structure Definition Language (DSDL) [8]. Schematron differs from the previous languages in the way it expresses the rules and constraints embodied in the schema. It uses a tree-like rule based system as opposed to the grammatical oriented constraint declaration. Inconsistencies are thus identified through the matching of patterns which encode the document rules.

3 Schemas and Grammars

A grammar is used to define, in a logical form, all the constraints contained in a schema. As we have seen before, the input is a tree-like object model to be subject of validation. Any type of grammar capable of express constraints relative to a tree-like model is suited to this task. One type of grammars capable of satisfying these requirements is **tree grammars**, which is the mechanism for describing permissible trees. Over the years many researchers used tree grammars for schema representation (DTD, WXS, RELAX NG developers and others).

Tree grammars can be divided into four main subclasses: local, single-type, restrained-competition and regular tree grammars [9]. A **local** tree grammar provides a single content model for all content of each element (terminal). A **single-type** tree grammar does not have this tight restriction, but cannot allow two production rules competing in the same terminal to have different content models. A **restrained-competition** tree grammar lifts this restriction. A **regular** tree grammar accepts any kind of production. In terms of expression and ordered from the more expressive to the less expressive (that is some grammars cannot be rewritten as other grammars) are

[2] http://www.xml.gr.jp/relax/
[3] http://www.thaiopensource.com/trex/
[4] http://xduce.sourceforge.net/

regular, restrained-competition, single-type and local tree grammars. DTD, WXS and RELAX NG are examples of respectively local, single-type and restrained-competition tree grammars.

Beside tree grammars, other grammars can be used, like **rule-based grammars** (rule-based systems). These grammars define assertions about the presence or absence of patterns in the document object tree. Schema languages using rule-based grammars are called rule-based schema languages (also called pattern-based or assertion-based) [10]. Schematron is an example of a schema language based on a rule-based grammar.

Table 1. Analysis of the schema language constraint.

Constraints		Element				Text				Attr				AttrValue			
		DTD	WXS	RNG	Sch	DTD	WXS	RNG	Sch	DTD	WXS	RNG	Sch	DTD	WXS	RNG	Sch
Lexicon	Support	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	Namespace Support	N	Y	Y	Y	-	-	-	-	N	Y	Y	Y	-	-	-	-
Syntax	Primitive Datatypes	-	-	-	-	N	Y	Y	Y	-	-	-	-	N	Y	Y	Y
	New Datatypes	-	-	-	-	N	Y	Y	N	-	-	-	-	N	Y	Y	N
	Mandatory	Y	Y	Y	Y	Y/N	N	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
	Cardinality	Y/N	Y	Y	Y	N	N	Y	Y	-	-	-	-	-	-	-	-
	Order	Y	Y	Y	N	N	N	Y	N	-	-	-	-	-	-	-	-
	Alternatives	Y	Y	Y	Y	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	Y
Semantics	Co-occurrence	N	N	N	Y	N	N	N	Y	N	N	N	Y	N	N	N	Y
	Relational	-	-	-	-	N	N	N	Y	Y	N	Y	Y	-	-	-	-

Legend: supported (Y); unsupported (N); incomplete support (Y/N); not applied (-).
Document Type Definition (DTD); W3C XML Schema (WXS); RELAX NG (RNG); Schematron (Sch).

4 Schema Constraints

In this paper the schema constraints expressed in grammars are classified in three different concepts or levels of abstraction. One is the **lexicon**, which deals with the domain vocabulary (identified XML entities); other is the **syntax**, that deals with structure *per se* of the document (tree grammars are specially focus at this part); and finally, the **semantics**, also called co-occurrence constraints, which deals with constraints relative to the content of the document. In this section, this paper evaluates and compares each schema language according to these three concepts. Table 1 resumes this evaluation showing the languages' support for each identified constraint type and the corresponding XML entity.

Note that each level requires that the previous constraints (belonging to the previous level) are matched; otherwise no further constraints can be evaluated. So if a document is said to be semantically correct, it means that it is also syntactically, and consequently lexically correct.

4.1 Lexicon

The schema language must include support for elements, attributes, attribute values and text, and also the support for namespaces.

In this subsection, this paper evaluates the capacity for the language to support the lexicon present in the tree object model. For validation purposes only a subset of the XML language is used. Things like parameter entities, processing instructions, DTD and CDataSections are processed and discarded when parsing the XML document. So the tree object model is reduced to four basic entities: **Elements**, **Attributes (Attr)**, **Attribute Values (AttrValue)** and **Text** (Comments are excluded since they do not represent relevant content information). The Figure 1 represents the basic entities that are involved in an XML document and their relations.

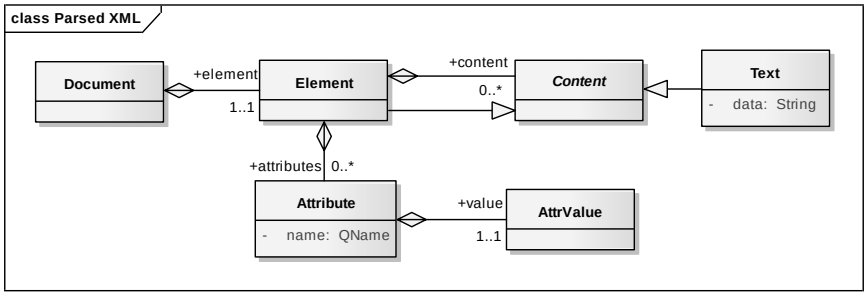


Fig. 1. Basic XML entities that are subject of validation.

All schema languages support the domain vocabulary contained in the model. WXS only recognizes text when it is the only content contained in an element (although it allows the declaration of mixed content it does not mandate where the text must appear when interleaved with elements). DTD recognizes text as PCDATA, and slightly lifts this restriction, enabling the definition of constraints with text content in between. RELAX NG and Schematron deal with text as any other kind of content. Concerning namespaces, only DTD does not have any support for this lexical entity. WXS supports namespaces naturally in the element and attribute declarations. RELAX NG allows at any time de selection of a namespace by using a specific statement (*nsName*, *name* or *anyName*). Schematron supports namespaces by relying in XPATH[5] expressions to construct expression patterns. XPATH expressions support namespaces naturally using the namespace prefix associated to the names of elements and attributes, and previously defined in the schema, using the natural XML definition of namespaces.

[5] http://www.w3.org/TR/xpath

4.2 Syntax

Syntax deals with the constraints related to the document structure. How the elements, attributes, attribute values and text are combined, placed and coded to form a meaningful environment of information.

In this kind of documents, syntax deals with the structuring of the tree; the alignment and leveling of elements, and also text, attribute placement and encoding. Tree grammars are specially focused on the definition of these constraints on tree structures. One can also use a rule-based grammar to represent structure related constraints that are missing in the tree grammar, particularly if we are using a less expressive tree grammar. Nevertheless, the use of rule-based grammars is not always straight forward and can be very hard to implement. DTD, WXS and RELAX NG are examples of respectively local, single-type and restrained-competition tree grammars. As expected RELAX NG, being a restrained-competition tree grammar, offers the more expressive approach to the representation of structure constraints. These constraints were classified in 5 different categories: **datatypes**, **mandatory**, **cardinality**, **order** and **alternatives** constraints.

Datatypes

DTD can only express the datatype of attributes in terms of explicit enumerations and a few coarse string formats. There has no support to describe numbers, dates, currency values, and so forth. Furthermore, DTD has no ability to express the datatype of character data. WXS fully supports datatypes by adding a rich set of primitive types and by enabling the declaration of new derived types from the existing primitive types. Unfortunately this is done only for attribute values but not to character data. Some authors [11] discuss the design principles that guided which types were to be defined and which were to be defined as primitives, accusing WXS for not having a well defined datatype hierarchy. RELAX NG offers a different approach to the problem. It firstly decouples the schema language from the set of datatypes, facilitating the evolution in an independent way of datatypes without compromising the language. Secondly it allows datatypes to be specified uniformly for both attribute values and element content; this is in accordance with the philosophy of uniform treatment for elements and attributes. RELAX NG introduces the concept of a Datatype Library, which provides a semantic model for a collection of data. Any collection of datatypes that can fit into the RELAX NG model can potentially be used as a RELAX NG datatype library, in particular, the datatypes defined by WXS Part 2 [4].

Mandatory and cardinality constraints

In DTD, the mandatory constraint is enforced by default simply by declaring the element in the parent element declaration. To declare cardinality of elements the operators '*' (zero or more), '?' (zero or one) and '+' (one or more) are used. For attributes the mandatory constraints is satisfied by the *REQUIRED* (no type for optional constraint) type defined in the attribute declaration. Text content can only be constrained to appear in the parent element content or not. In WXS the mandatory constraint is enforced by the declaration of elements and attributes. For attribute values this constraints is satisfied by defining a datatypes for the attribute. Cardinality

constraints for elements are enforced using the *minOccurs* and *maxOccurs* statements in particle statements (*Element*, *Choice*, *Sequence*, *Any*, *All* and *Group* statements). There are no mandatory and cardinality constraints for Text content. Like WXS, in RELAX NG, the mandatory constraint is enforced by the declaration of an element, text, attribute or value for an attribute. Cardinality constraints are partially satisfied by the *ZeroOrMore*, *OneOrMore* and *Optional* statements. As opposed to WXS, RELAX NG does not support the declaration of explicit and exact cardinality using integer boundaries. Schematron uses counting operations present in XPATH expressions to enforce mandatory and cardinality constraints. These counting operations can be applied to any type of content. Schematron relies on XPATH expressions to pattern matching and therefore is limited to the language ability to define types. XPATH only recognizes the numeric and string types but, on the other hand, can enforce these types at any type of content.

Order constraints

In DTD, order constraints for elements are supported by the ordering of elements in the children declaration. Nevertheless it stops being supported when mixed content is defined (Text and elements as content). In WXS the order of elements are defined by *Sequence* statement mandating that the order of declaration is the order in the document structure. No order for text content can be established. In RELAX NG the order of the statements and declarations mandate the order on the document. Text can be constrained anywhere in the document content just by placing the *Text* statement. Schematron can support these constraints by checking positions, in XPATH expressions, of elements or text content in the parent element. This may lead to incredible large expressions and are difficult to combine with alternative and cardinality constraints.

Alternative constraints

In DTD alternative constraints for elements are supported by the operator '|' (choice) and for attribute values with the enumeration of available datatypes (very few). For text and attributes there is no support for this kind of constraints. In WXS alternative constraints for elements are enforced by the *Choice* statement and can be combined with order constraints. Alternative for attribute values is defined with datatypes by using the *Union* (for type composition) and *Pattern* (for definition of character strings using regular expressions) statements. Like DTD, WXS does not support constraints for text and attributes. In RELAX NG, these constraints are satisfied by the *Choice* and *Interleave* statements. These statements can appear at any time in the schema enabling it to be applied to any type of content. This feature enables alternatives affecting more than one level in the object tree, making it possible to have alternative structures (skeletons). This is only possible because RELAX NG is a restrained-competition tree grammar. For AttrValues, alternatives can also be satisfied by the Datatype Library. Schematron again uses Boolean operations present in XPATH expressions to enforce alternatives to any given scenario, and therefore can be applied to any type of content. Nevertheless, this is not an immediate and simple way of defining alternatives.

4.3 Semantic constraints

Semantic constraints, also called **co-occurrence** constraints, are constraints between two or more values (content information). For example, "if an element has attribute A, it must also have attribute B" or "if an element has a parent X, then it must have an attribute Y". A co-constraint may exist between any kind of entity in the content model (elements, attributes, attribute values, and text). The occurrence of these constraints in the document model may lead to the identification of missing, illegal, duplicate, unordered, misplaced content. This kind of constraints is difficult or impossible to define using tree grammars (their primary focus is structure) and is commonly satisfied by rule-based grammars. In this comparison, Schematron is the most suited (sometimes the only) language to represent this kind of constraints. Schematron achieves this by being able to represent **co-occurrence constraints** using patterns coded in XPATH paths and expressions. Neither DTD, WXS and RELAX NG can. Some considerations about validation using rule-based constraints and particular the XPATH ability to express them are discussed in [12].

In this paper, a new subgroup of co-occurrence constraints is added, the **relational constraints**. These are a particular kind of co-occurrence constraints that deal with relational concepts like uniqueness, keys (identity) and key-references. These concepts allow systems to apply normalization according to the normal forms to schemas and benefit from them [13]. DTD (ID for identification and IDREF for key-references) and WXS (*Unique* for uniqueness, *Key* for keys and *KeyRef* for key-references) have explicit statements responsible for assuring this kind of constraints. Nevertheless, DTD only supports keys and key references for attributes and only one at a time. DTD has no support for the uniqueness constraint. On the other hand, WXS supports relational constraints for any kind of XML entities including combinations of them (e.g. a key composed of two separate attributes). RELAX NG has no support for relational constraints. Schematron can support these constraints making use of the search capacity of the language to assure uniqueness and existence of the keys.

5 Language Characteristics

In this section some of the main characteristics of the languages under analysis are discussed. Table 2 gives an overview of these characteristics, ordering each schema language from the most compliant with the feature to the less.

Expressivity

Expressivity is the measurement of the constraints enforced by the schema language. The more constraints a schema language allows the more expressive the language is. DTD is the less expressive language studied. It offers a limited set of statements allowing the enforcement of only a few constraints. Although the constraints available are able to cover most of schemas constraint requirements, it lacks the ability to define other, more refined constraints. Even though some constraints require a little more work for the developers to be defined, WXS offers a greater set of statements enabling the declaration of many constraints. One important

disadvantage of this language is the inability to deal with character data as typed content like RELAX NG. The source for the RELAX NG capacity to express constraints is the ability to apply the same statements to many different contexts. Being a restrained-competition tree grammar also contributed to wider the number of constraints being enforced. Schematron is the most expressive language of all, allowing the enforcement of almost all the constraints identified for a document.

Reusability

Reusability is the ability of a language to pack and reapply sets of statements in different contexts. DTD has no immediate way of reusing statements in the schema (only by entity replacement of attributes). Reusability in WXS is satisfied with the construction of types and attributes groups and their assignment respectively to elements and attributes. Schematron enables reusability by defining abstract patterns and rules and instantiating (with the ability to define parameter variables) them several times in the document. RELAX NG takes reusability a little bit further, allowing the definition of sets of any type of statements and reapply them to any context in the schema.

Compactness

Compactness is the ability to build the most constraints possible using the less available statements. Schematron produces very long and complex schemas due to the pattern-based approach. Schematron should be avoided when enforcing of lexical and syntactic constraints are required due to its non-structure approach. Schematron should only be used when the other languages are incapable of enforcing a constraint. WXS produces extensive schemas due to the leverage of the tree structure in a list of element declarations and the added complexity of type construction. Type construction in WXS requires an elaborate set of statements to define extensions and restrictions on content making it very extensive. With WXS the more complex the schema is the more extensive the schema becomes[6]. RELAX NG also produces very compact schemas. This is achieved by the ability of RELAX NG to follow the natural tree structure of the document combined with the huge versatility of the statements to be applied in wider contexts. Another important feature is the ability of RELAX NG to collect all types of statements and reapply them at any time and any context. Although the DTD language also leverages the document tree structure (as for WXS), it is able to produce very compact schemas given the universe of constraints possible of being enforced. The reduced expression of this language and therefore, the few statements available contributed to the compactness of the schema. The more constraints we need to enforce more difficult becomes the task of building the language primitive statements.

Complexity

Complexity is the level of knowledge required to the correct development of a schema. The reduced expression of the DTD language and the straight forward approach used in element and attribute declarations, make it very easy to understand.

[6] A more detailed comparative analysis of the language vocabulary present in WXS and RELAX NG is presented at: http://www.wyeast.net/compare.html

WXS is the most difficult language. Its approach to object oriented design makes it difficult to use and understand. Although very powerful, the type construction is not immediate for unskilled developers. The extension and restriction mechanisms are also difficult to understand and require previous knowledge and experience before using them, especially when dealing with attributes. An extended evaluation of the complexity of this language and best practices for its usage are discussed in [14]. RELAX NG is a very simple language. Again its ability to apply the same statements to many different contexts; reduced the schema vocabulary and therefore reducing the knowledge needed to develop a schema. Another aspect of RELAX NG that contributed to the reduced complexity of the schema is the ability of the language to follow the document tree structure, making it easy to develop a schema only by looking at the expected document. Some features and design characteristics of this language are discussed in detail in [15]. Again the rule-based approach of Schematron makes this language difficult to learn and even more difficult to understand the need expression for the enforcing of a given constraint

Extensibility

Extensibility is a design principle to take into consideration the language future growth. It is reflected in the ability of the schema language to add new features (increments), be enhanced or customized, without disturbing the existing features. Languages that use XML syntax for schema notation increase the ability to be extended. This is achieved by the ability of XML to be annotated with elements and attributes from other namespaces.

WXS provides a special element for this purpose (*AppInfo*), enabling the composition with other schema languages, especially Schematron [16]. Although RELAX NG does not provide specific elements or attributes to this purpose, it has an open syntax predicting this requirement. Schematron as a XML syntax language also allows schema extension. Nevertheless one of its main design principles was to be combined with other existing schema languages and not to import. Some interesting work in this area can be read in [17]. DTD has no mechanism for schema extension, mainly because it was the first effort in schema languages design. At the time DTD developers did not took this aspect in consideration because they did not predict the existence of other schema languages.

Table 2. Languages ordered by their relative characteristics.

	most	>>	>>	**less**
Expressivity	Sch	RNG	WXS	DTD
Reusability	RNG	Sch	WXS	DTD
Compactness	DTD	RNG	WXS	Sch
Complexity	Sch	WXS	RNG	DTD
Extensibility	WXS	RNG	Sch	-

Legend: Document Type Definition (DTD); W3C XML Schema (WXS); RELAX NG (RNG); Schematron (Sch).

6 Other Related Work

This paper combines the work of many different papers. It follows some of the guidelines present in [18] but introduces a different classification of the constraints being supported in each schema language. The approach used in this paper was oriented to the grammar used by the schema language to encode the schema constraints. This approach was inspired in the work of [9] based on a taxonomy of XML Schema Languages but introducing the rule-based grammar approach. Other works like [19] presented a rapid overview of the state of art of schema languages.

7 Conclusions and Future Work

As a result of this work some valuable conclusions were reached to anyone willing to formalize schemas for XML documents (some conclusions converge with the concepts guiding the DSDL project [8]).

- The schema language should be chosen carefully in order to define the most constraints possible using it. In this sense always:
  - Use a tree grammar for the definition of lexical and syntactic constraints. Consider always the most expressive tree grammar possible like restrained-competition grammars (RELAX NG is a good choice). When dealing with schemas requiring only the definition of simple vocabulary, consider the use DTD (or any other tree grammar based language).
  - Avoid using rule-based grammars as preferred schema language. Otherwise the schema becomes very extensive and complex.
  - Choose the schema language that adequately satisfies your datatype constraints. Otherwise, additional constraints will be required to restrain datatype values. Consider using a language that supports importing datatype libraries (e.g. WXS Datatype Library) like RELAX NG.
- Consider using additional schema languages when the chosen schema language does not support all the schema constraint requirements.
  - Use rule-based grammars when no tree grammar is able to satisfy the needed requirements. Otherwise the schema becomes very extensive and complex. Schematron is a good candidate since it can easily be embed in other schema languages and offers a great expressivity in defining schema constraints.
- Finally, in order to ensure that schemas remain simple and compact, always remind to choose the most appropriate schema language statement even if it means importing from other schema languages. WXS and RELAX NG offer a good choice for importing language statements for specific constraints.

For future work, it would be interesting to study the problems that users frequently face when developing schemas and provide possible solutions looking at some of the existing schema languages.

References

1. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, John Cowan. *Extensible Markup Language (XML) 1.1 (Second Edition)*. W3C, September 2006. See http://www.w3.org/TR/xml11/.
2. ISO 8879:1986 Information processing - Text and office systems - Standard Generalized Markup Language (SGML)
3. Fallside, David C. and Walmsley, Priscilla, editors. *XML Schema Part 0: Primer Second Edition*. W3C, October 2004. See http://www.w3.org/TR/xmlschema-0/.
4. Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, editors. *XML Schema Part 1: Structures Second Edition*. W3C, October 2004. See http://www.w3.org/TR/xmlschema-1/.
5. Biron, Paul V. and Malhotra, Ashok, editors. *XML Schema Part 2: Datatypes Second Edition*. W3C, October 2004. See http://www.w3.org/TR/xmlschema-2/.
6. Clark, James and Mokoto, Murata. *RELAX NG Specification*. OASIS, December 2001. See http://www.oasis-open.org/committees/relax-ng/spec.html.
7. *Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation – Schematron*. ISO/IEC 2004. See http://www.schematron.com/iso/dsdl-3-fdis.pdf.
8. ISO/IEC 19757-1. *Document Schema Definition Languages (DSDL) – Part 1: Overview*. November 2004. See http://dsdl.org/0567.pdf.
9. Murata, Makoto, Lee, Dongwon, Mani, Murali. Taxonomy of XML Schema Languages using Formal Language Theory. ACM Transactions on Internet Technology (TOIT), Volume 5, Issue 4 (November 2005), pp. 660-704. See http://doi.acm.org/10.1145/1111627.1111631.
10. Dodds, Leigh. *Schemarama*. XML.com, 7 February 2001. See http://www.xml.com/pub/a/2001/02/07/schemarama.html.
11. Lewis, Amelia. *Not My Type: Sizing Up W3C XML Schema Primitives*, Julho 2002. See http://www.xml.com/pub/a/2002/07/31/wxstypes.html.
12. Provost, Will. *Beyond W3C XML Schema*, Abril, 2002. http://www.xml.com/pub/a/2002/04/10/beyondwxs.html.
13. Provost, Will. *Normalizing XML, Part 1*. Novembro 2002. http://www.xml.com/pub/a/2002/11/13/normalizing.html.
14. Obasanjo, Dare. *W3C XML Schema Design Patterns: Avoiding Complexity*. Novembro, 2002. See http://www.xml.com/pub/a/2002/11/20/schemas.html.
15. Clark, James. *The Design of RELAX NG*. See http://www.thaiopensource.com/relaxng/design.html.
16. Costello, Roger L. *Extending XML Schemas*. March 06, 2001. See http://www.xfront.com/ExtendingSchemas.html.
17. Robertsson, Eddie. *Combining Schematron with other XML Schema languages.* Junho, 2002. See http://www.topologi.com/public/Schtrn_XSD/Paper.html.
18. Lee, Dongwon, Chu, Wesley W. Comparative Analysis of Six XML Schema Languages, June 7, 2000. See http://www.cobase.cs.ucla.edu/tech-docs/dongwon/ucla-200008.pdf.
19. Rick Jelliffe. *The Current State of the Art of Schema Languages for XML*. 2001. See http://www.planetpublish.com/pdfs/RickJellif.

schem@Doc: a web-based XML Schema visualizer

José Paulo Leal[1] and Ricardo Queirós[2],

[1] CRACS & DCC-FCUP, University of Porto, Portugal
zp@dcc.fc.up.pt
[2] CRACS & DI-ESEIG/IPP, Porto, Portugal
ricardo.queiros@eu.ipp.pt

Abstract. XML Schema is one of the most used specifications for defining types of XML documents. It provides an extensive set of primitive data types, ways to extend and reuse definitions and an XML syntax that simplifies automatic manipulation. However, many features that make XML Schema Definitions (XSD) so interesting also make them rather cumbersome to read. Several tools to visualize and browse schema definitions have been proposed to cope with this issue. The novel approach proposed in this paper is to base XSD visualization and navigation on the XML document itself, using solely the web browser, without requiring a pre-processing step or an intermediate representation. We present the design and implementation of a web-based XML Schema browser called schem@Doc that operates over the XSD file itself. With this approach, XSD visualization is synchronized with the source file and always reflects its current state. This tool fits well in the schema development process and is easy to integrate in web repositories containing large numbers of XSD files.

Keywords: Schema visualization, XML Schema, Transformation, Documentation, Interoperability.

1 Introduction

XML Schema is arguably the most used type definition language for XML documents. Thus, developing new XML documents types usually involves reading and understanding its definition in XML Schema (XSD). The teams that develop these definitions usually integrate persons with different backgrounds, including experts in the domain and experts in XML Schema. Typically, domain experts are not familiar with the complex features of this specification (e.g. reference bindings, extensions, restrictions and redefinitions). To make things harder for the all team, an XSD is serialized as an XML document, which makes it possible but difficult to read by humans. As postulated by XML design goals [1], although XML tags are usually self explanatory, and that is certainly the case with XML Schema, the resulting XML documents are usually anything but concise. This is not a problem for software processing XSD files, but is an extra burden for a person reading them in detail.

In this paper we present the design and implementation of a web-based tool for browsing XML Schema files called schem@Doc. Using this tool a human reader browses XSD files through a Web interface, using tree structures, tables and formatted text, rather than reading directly the source code. The schema structure and documentation are automatically generated from the document type definition, helping users to visualize the relationships among type definitions and reading documentation in plain natural language text formatted in XHTML. This tool does not require specialized knowledge of XML or XSD, so it can be used by novice users. Also, it does not require any pre-processing of XSD files. The user needs only to open an XSD file annotated with a special processing instruction on a conventional Web browser.

This tool was designed for Web based systems requiring the visualization of XSD files still under development, by users that may be unfamiliar with the XML Schema specification. Since it is a very light tool, it is well adapted for repositories with large numbers of XSD files. Although targeted primarily for interactive use, it also produces printed versions of XSD files that can be used as reference.

The remainder of this paper is organized as follows. The next section reviews basic concepts related to schema definition languages and existing approaches to browse schema definitions. Section 3 present an overview of schem@Doc and section 4 details its main features such as the navigation and visualization of the schema types, embedded Schematron support and example generation using instance documents. The last section draws conclusions from the presented work and outlines future developments.

2 Visualization of schema definitions

Document Type Definition (DTD) was the language inherited from SGML, to define types of documents in XML. Its many limitations [2] (e.g. insufficient data type support, lack of namespace awareness) lead to an official W3C recommendation for a schema language called XML Schema Definition language [3] (XSD) in 2001. XML Schemas are richer and more powerful than DTDs [4] and are written in XML. This new language overcame DTD limitations and provided several advanced features, such as the ability to build new types derived from basic ones [5], manage relationships between elements (similar to relational databases) and combine elements from several schemata.

In spite of its expressiveness, XSD lacks features to describe constraints on the XML document structure. For instance, there is no way to specify dependencies between attributes, or to select the content model based on the value of another element or attribute. To address these issues several schema languages were proposed, such as RELAX NG [6] (based on TREX [7] and RELAX [8]), DSD (Document Structure Description) [9] and Schematron [10]. The Schematron language provides a standard mechanism for making assertions about the validity of an XML document using XPath expressions and can be easily combined with W3C XML Schema documents.

These schema languages have different syntaxes but most of them support XML serialization. It is a well known fact that the XML formalism was not designed for the convenience of human readers [1]. Although tag and attribute names are usually self explanatory and they are meaningful to an expert, large documents with complex structure are difficult to understand and require specialized viewers. Specially for XSD files, several tools are available for browsing and displaying schema definitions in a user-friendly way. These tools are usually part of commercial XML Integrated Development Environments (IDE) such as XML Spy, Stylus Studio and <oXygen/> Oxygen, or plug-ins of general programming IDEs such as Eclipse, NetBeans or even Visual Studio .NET. There are a few standalone tools [11, 12, 13] designed to browse and generate documentation from XSD files. These approaches require either a preprocessing of the XSD files (converting them to HTML files), or installing specific software for browsing them. On the other hand, the majority of these systems are not open source, which limits its use in non-commercial research projects.

3 Overview

XML documents play such an important role in publishing and exchanging data on the Web that they might be considered a universal language tool [14]. The design of the schem@Doc tool fits the ubiquity of XML documents and it is easy to install on a web server and use from any browser linked to the Internet. Simplicity is the key in the design of this tool, since it is the best way to ensure reliability and efficiency. Hence, it should be no surprise that our schema viewer is composed of the following three components:

a transformation for converting an XSD to a web layout;
a script for handling user interaction;
a stylesheet to configure the layout.

The interplay of these three components to produce an interactive visualization from a XSD file is represented schematically in Fig. 1.

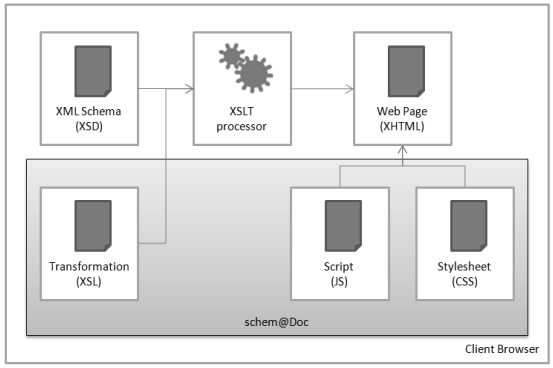


Fig. 1. Overall architecture of schem@Doc.

A Web browser is the only requirement for using the schem@Doc tool. The XSD file is opened from the Web browser and processed as a regular XML file. We rely on the XSLT processor, included in all recent versions of standard Web browsers, for the transformation of the XSD file into a web layout. The XSLT processor loads an XSLT transformation and produces a XHTML web layout that will in turn load the script and stylesheet files of the schem@Doc package.

To prepare an XSD file for schem@Doc it is only necessary to annotate it with a Processing Instruction (PI) in its prologue, to activate the XSLT processor and load the transformation file. The following code shows a PI in a schema file prologue:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="schemaDoc.xsl"?>
<xsd:schema …>
…
</xsd:schema>
```

The transformation file, as well as the script and stylesheet files it requires, are all part of the schem@Doc distribution package available for download from the following URL http://www.dcc.fc.up.pt/schemaDoc. In a typical installation these files are placed on web published directory and shared by several XSD files.

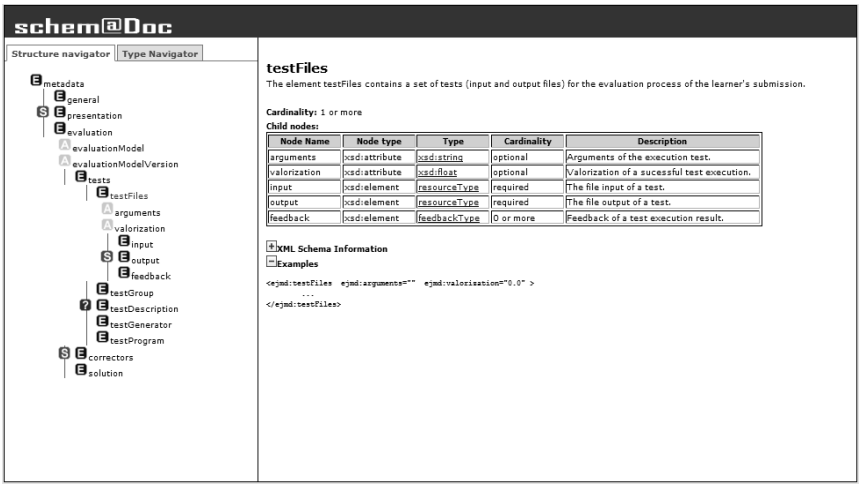


Fig. 2. XSD displayed by schem@Doc.

When an XSD file with the above PI is opened on a Web browser the user will be presented with an interface similar to the one depicted in Fig. 2. Of the left part the reader finds a navigation area with two tabs, each proving its own way to browse and select types defined on the XSD. Documentation on selected types is displayed on the right part. These features of schem@Doc are described in greater detail in the next section.

4 Main features

Schem@Doc is a Web-based tool to navigate through the types defined on a W3C XML schema documents and to view their documentation. The remainder of this section details its navigation, visualization and example generation features.

4.1 Navigation

When designing the web interface of schem@Doc we considered two distinct usage profiles. Novice or first-time users are expected to use schem@Doc to understand the structure of an XSD and identify its main types. Expert and recurrent users, that already know the basic structure of the XSD, are expected to search for documentation on specific schema type, to focus on technical details and may need to print a reference for off-line usage. To address this range of needs this tool includes two navigation modes, depicted side by side in Fig. 3.

Structural: browse the schema file based on the structure of instances;
Reference: list of complex and simple types in alphabetic order.

The former is the default and displays the structure of a valid XML instance document, with elements, attributes and respective types as tree nodes. The roots of these structures are element definitions on the XSD. Type definition referred by other types (using attributes `type`, `ref` or `base`) will appear as a child of the referring type in the hierarchy. These nodes can be expanded and collapsed to control visualization.

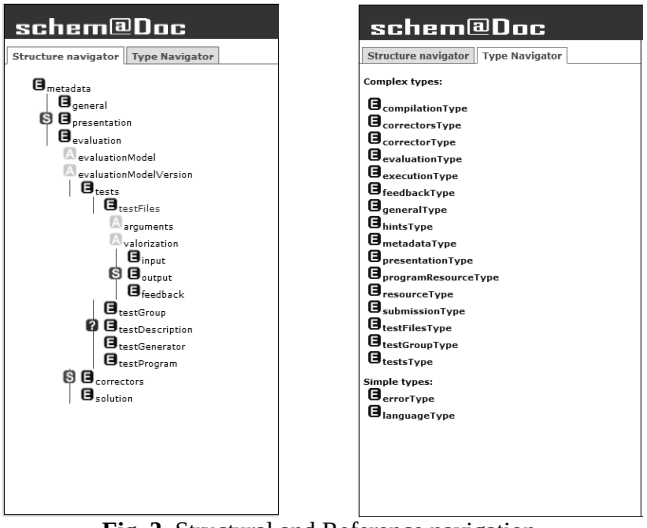


Fig. 3. Structural and Reference navigation.

The Reference navigation provides direct access to specific types and their documentation. This list is split up in complex and simples types and sorted in alphabetic order within each group.

Both navigation modes support the visualization of schema definitions loaded using the W3C XML Schema elements: import and include.

4.2 Views

After selecting an element/attribute in the navigation area (left side), the user visualizes automatic generated documentation related to that type on the viewing area in the right part of the screen. Fig. 4 shows the visualization presented to the user after selection of the `metadata` element on the structure navigator.

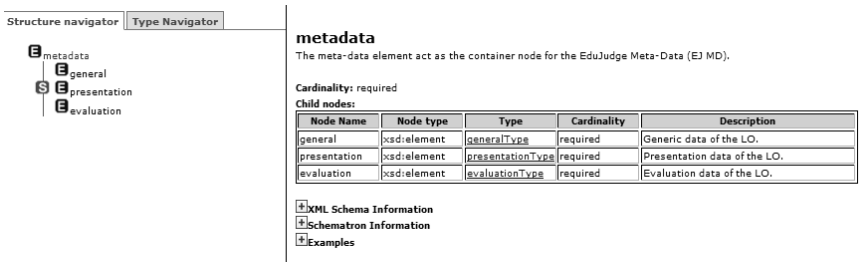


Fig. 4. Visualization of type documentation.

The generated documentation contains several sections, namely: element/attribute name, description, cardinality, a table with summary on child nodes information. It contains also technical information on schemata (both XSD and embed Schematron, presented in following sub-section) and usage examples. These last sections are meant for the expert user thus are initially presented collapsed.

As mentioned before, XSD files can be hard to understand and undocumented ones are even harder. To mitigate this problem the W3C specification includes the `annotation` element for adding complementary information. This element can annotate any XSD elements and has two child elements: `appinfo` and `documentation`. These elements can occur zero or more times inside the `annotation` element. The schem@Doc tool uses the `documentation` element to generate description for types.

The annotations allowed by the W3C specification are not enough for the needs of schem@Doc. For that purpose, we created a new document type, with a target namespace that separates new elements and attributes from the schema itself. It includes elements and attributes to define and control the automatic generation of the schema documentation. The following code illustrates the use of the type attribute to classify the descriptions found in the documentation element either as "extensive" or "summary". Note that these examples assume a prior namespace declaration associated with "doc" prefix.

```
<xsd:element name="general" type="generalType">
  <xsd:annotation>
    <xsd:documentation doc:type="description-extensive">
      Extensive description here.<xhtml:br/>
    </xsd:documentation>
    <xsd:documentation doc:type="description-summary">
      Summary description here.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

The previous example shows also how XHTML tags can be used to format the schema elements descriptions, thus enriching the readability of the documentation. According to the W3C XML Schema specification `documentation` element may contain children from any namespaces. Since the content of this element is injected in a XHTML template, elements of this namespace are particularly recommended when using a XSD with schem@Doc. Typical XHTML elements used in this context are hypertext references to HTML pages with extended documentation, images containing diagrams and even tables to improve the documentation content layout.

4.3 Support for embedded Schematron

In spite of its expressiveness, there are cases where it is impossible to constraint an XML document instance using XML Schema, as in the following examples:

- Specify incompatible/complementary attributes;
- Force an element or attribute value to be greater/less than other;
- Make the content model depend on the value of an element or attribute.

To validate this type of constraints schema authors usually extend XSD files using at least one of these options: 1) combine XML Schema with others schema languages; 2) write code in a programming language to express the additional constraints; 3) use an XSLT/XPath stylesheet. To avoid ad-hoc solutions in schem@Doc we opted for combining XSD with other standard schema languages.

A usual candidate to perform this "second level of validation" is the rule-based validation language Schematron. The combination of the Schematron rules with the W3C schema can be done in two ways: as separate files, using pipeline validation languages such as the DSDL (Document Schema Definition Language) or Schemachine language, or as a single file, in this case embedding Schematron rules in the XML Schema using the `appinfo` element within the `annotation` element of a particular XSD element. This last approach fits our processing model best and thus was selected.

The following example shows an XSD that includes Schematron rules to define constraints that are beyond the capabilities of XML Schema. Specifically the pattern enforces the existence of a `time-submit` element if a `time-solve` element also exists.

```
<xsd:appinfo>
 <sch:pattern name="p2">
  <sch:rule context="time-solve">
   <sch:assert test="following-sibling::*[1]= time-submit">
If the element time-solve appears in the document, then the
timesubmit must appear after.</sch:assert>
  </sch:rule>
 </sch:pattern>
</xsd:appinfo>
```

Fig. 5 shows how schem@Doc renders this particular rule in the viewing part. It should be noted that Schematron information is placed inside a special section that is initially collapsed. As for XSD, Schematron documentation is automatically generated from its XML source.

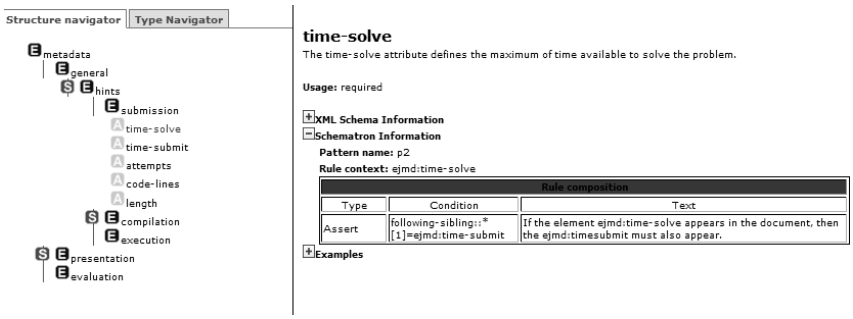


Fig. 5. Visualization of Schematron rules.

4.4 Visualization of instance examples

The automatically generated documentation includes an examples section to illustrate the use of a type with a fragment of a document instance. Surely, the XSD author can include these fragments in XSD `documentation` elements and even format them using XHTML. This special type of documentation could have been just another value in the `type` attribute introduced in sub-section 4.2. However, producing examples this way would be time-consuming and error-prone.

To produce document instance fragments as example for particular elements or attributes, schem@Doc uses a set of instance documents defined by the XSD author. To generate an example fragment for documenting an element definition, the tool simply searches the document instances for occurrences of those elements. These instance files may be created from the XSD file itself using an IDE (e.g. Eclipse) or a XML specialized editor (e.g. XESB [15]). The visualization of the fragment example is controlled by using the schem@Doc namespace at two levels. At the root level it uses the `href` attribute to identify document instance(s). At the element level it uses the `order`, `childDepth` and `parentDepth` attributes to control, respectively,

what document instance should be used and the depth level visualization of the example. The following example illustrates the use of these attributes to control the visualization of examples for a particular element named `presentation`.

```
<xsd:schema doc:href="ex1.xml ex2.xml" …>
   <xsd:element name="presentation" doc:childDepth="0"
   doc:parentDepth="1">
      <xsd:annotation>
        <xsd:documentation>…</xsd:documentation>
      </xsd:annotation>
   </xsd:element>
</xsd:schema>
```

The `childDepth`/`parentDepth` attributes defines how many ancestors and descendants levels should be shown in the example. The absence of the `order` attribute, at the element level, will make the tool to iterate over all instance documents included in the root level `href` attribute, until it finds the first occurrence of the `presentation` element. Fig. 6 shows how schem@Doc renders the example fragment.

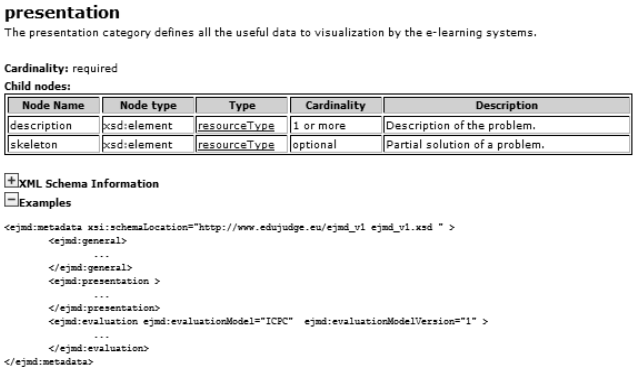


Fig. 6. Visualization of fragment examples.

5 Conclusions and future work

In this paper we described the design and implementation of a Web-based XML Schema documentation generator called schem@Doc. The main contributions of this work are a new approach to the visualization of schema definitions and a tool to present XML Schema files on the Web. The schem@Doc tool provides direct visualization of the XSD files, generating documentation on-the-fly from the XML schema file itself, without requiring the installation of add-ons or plug-ins at the client side. The tool is being distributed in open source, and is available for testing and download at the following URL `http://www.dcc.fc.up.pt/schemaDoc`.

Although the current implementation is already being used for documenting XSD files developed as part of an eLearning project, it has some limitations that we will address in the near future:

- Ensure full support for the XML Schema specification (e.g. field, key, keyref, redefine, selector, unique elements and abstract, substitutionGroup attributtes);
- Handle recursive definitions in the navigation view;
- Improve the initial responsiveness by generating type information on a lazy basis .

Make extensions of schem@Doc compatible with existing editors to avoid the need to edit the XSD source code. After addressing these limitations we plan to evaluate the this tool experimentally, testing both its usability and utility with users with different knowledge levels of XML Schema . We plan also to improve the support to printed versions of XSD files that can be used as reference and develop a search feature based on XPath expressions.

Acknowledgements

This work is part of the project entitled "Integrating Online Judge into effective elearning", with project number 135221-LLP-1-2007-1-ES-KA3-KA3MP. This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

References

1. Birbeck, M. And others: Professional XML, Wrox, 2ª edição, 2001.
2. Harold, E.R.: The XML Bible, 3rd edition, Hungry Minds, 2004.
3. Fallside, D.C. (Ed.): XML Schema Part 0: Primer Second Edition. World Wide Web Consortium, Recommendation, 28 October 2004.
4. Song, G., Zhang, K.: "Visual XML Schemas Based on Reserved Graph Grammars," itcc, vol. 1, pp.687, International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 1, 2004.
5. Biron, P. V. and Malhotra, A. (ed.). XML Schema Part2: Datatypes Second Edition. W3C Document, October 2004.
6. Clark, J, Murata, M.: RELAX NG Specification, OASIS Committee Specification, December 2001, http://relaxng.org/spec-20011203.html
7. Clark, J.: TREX - Tree Regular Expressions for XML. Thai Open Source Software Center, 2001, http://www.thaiopensource.com/trex/.
8. Murata, M.: RELAX (Regular Language description for XML). INSTAC (Information Technology Research and Standardization Center), 2001, http://www.xml.gr.jp/relax/.
9. Moller, A.: Document Structure Description 2.0, BRICS, 2002, http://www.brics.dk/DSD/dsd2.html
10. The Schematron, An XML Structure Validation Language using Patterns in Trees, http://www.ascc.net/xml/resource/schematron/schematron.html
11. DocFlex/XSD Home Page: http://www.filigris.com/products/docflex_xsd/

12. Kilkelly, F.: SchemaViewer1.0, Internet Document, Nov. 2002. http://xml.coverpages.org/SchemaViewer10-Ann.html.
13. VisualSchema Home Page: http://www.visualschema.com/
14. Mignet, L., Barbosa, D., Veltri P.: The XML Web: a First Study, International World Wide Web Conference. Proceedings of the 12th international conference on World Wide Web, 2003.
15. Queirós, R., Leal, J.P.: Xesb: um editor XML para as massas. In José Carlos Ramalho, Alberto Simões e João Correira Lopes (eds.), In Actas da 3a Conferência Nacional XML: Aplicações e Tecnologias Associadas, (XATA 2005), Universidade do Minho, Braga, 2005.

GuessXQ, an inference Web-engine for querying XML documents

Daniela da Cruz[1,3], Flávio Xavier Ferreira[1,2,4], Pedro Rangel Henriques[1,5], Alda Lopes Gancarski[2,6], and Bruno Defude[2,7]

[1] University of Minho, Department of Computer Science, CCTC, Campus de Gualtar, Braga, Portugal
[2] Institut TELECOM, TELECOM & Manangement SudParis, CNRS SAMOVAR 9 rue Charles Fourier, 91011 Évry, France
[3] danieladacruz@di.uminho.pt
[4] flavioxavier@di.uminho.pt
[5] prh@di.uminho.pt
[6] Alda.Gancarski@it-sudparis.eu
[7] Bruno.Defude@it-sudparis.eu

Abstract. To search for specific elements in a marked up document we have, at least, two options: XPath and XQuery. However, the learning curve of these two dialects is high, requiring a considerable level of knowledge. In this context, the traditional *Query-by-example* methodology (for Relational Databases) can be an important contribute to make easier this learning process, freeing the user from knowing the specific query languages details or even the document structure.
In this paper, we describe how we implement *Query-by-example* in a Web-application for information retrieval from a collection of structured documents, the GuessXQ system. In essence, we built an engine capable of deduce, from a specific example, the respective XQuery statement. After inferring the generic statement, the engine applies it to all documents in the collection to perform the desired retrieval.
A suitable interface allows the end-user to mark over a sample document, picked up from the collection, the path he wants to select.

1 Introduction

In Structural Document Retrieval [3] the creation of a query that yields valid results strongly depends on the user-friendliness of the search engine interface. As structured queries are powerful but complex to write (the user must have a deep knowledge of the query language as well as the document schema), some specialised editors have been developed to ease this task (XMLSpy[2], EditiX[8],oXygen[9]).

"Example is always more efficacious than precept." This statement, by Samuel Johnson, led HCI[10] researchers to suggest a new interaction paradigm called

[8] http//www.editix.com
[9] http//www.oxygenxml.com
[10] Human-Computer Interaction.

Query-by-example (QBE). Born in the context of database querying [5], typical QBE systems are based on the "fill in the blanks" approach. QBE is based on the concept that the user formulates his query by filling in the appropriate skeleton tables the fields and/or restrictions on fields (the relational selection concept) he intends to search for. We adapt this approach to XML search by allowing to select and restrict entire paths (XML elements) directly on a sample document. There are some other works [1,4,6] which adapt the relational QBE model by showing the XML Schema Definition (XSD) tree instead of the table skeleton. Our system, not only displays the XML Schema tree representation to the user, but also an sample document from the collection. Elements selection and restriction is, then, directly done in the sample document, giving the user a complete indication about the information he is searching for.

From the selected paths, our engine, called GuessXQ, infers the complete query. After building the query, it is applied to all documents of the same type (schema) in the specified collection. The results are shown in a user-friendly Web interface used to select the referred path. The next section presents the GuessXQ system, followed by our conclusions.

2 GuessXQ system

The architecture of our QBE system, the GuessXQ engine, is presented in Figure 1. As depicted, the system is composed by the several modules.

The **Repository** is a collection of XML files grouped by their schema (XSD). This Repository, in a simple way, is composed by three tables: `XMLdocs`, `XSDfamilies` and `Relations`. The `XMLdocs` table stores the name of the XML document and its location; the `XSDfamilies` table stores the name of each XSD and its location; and at last, the `Relations` table relates each XML document with the XSD family where it belongs.

The **Repository access interface** allows the other modules to access the repository in a systematic and simple way. It allows the other modules to select a Schema, a collection of documents, a single document or a path in the document. This module is composed by a set of *fetch* methods for retrieving documents from the repository. This module works as an abstraction layer over the repository, allowing the system to change the repository paradigm in the future without a major architecture modification.

The **Interface module** is a GUI responsible for the interaction between the user and the system, allowing the user to set the "example" for the QBE engine. The user starts by choosing a document type (XSD) from the repository. Then, a document belonging to this family is suggested to him. By now, the sample document is selected to be of average size, and to contain the major number of elements/attributes. However, the user can change it, choosing a more significant one from the query perspective (i.e., an XML document that allows to perform more complex or complete queries).

Query specification includes the selection of components (elements or attributes) and the possibility to restrict them to the respective value in the sample

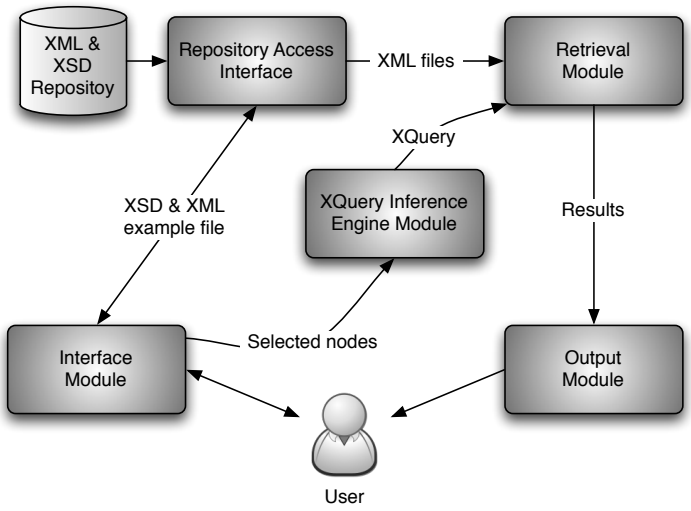


Fig. 1. GuessXQ architecture

document. For example, in Figure 2, the user specifies a query selecting elements (in yellow) and content values (in blue). This selection mechanism is aided by a table that maps each component or value with its corresponding XPath expression. After query specification, the interface module sends to the XQuery Inference Engine module the list of selected nodes and also the referred table, for query generation.

The **XQuery Inference Engine** module has the task of inferring the XQuery query, using the information sent by the Interface module.

The **Retrieval module** is responsible for the query execution. The query is executed in each document belonging to the selected schema, and the results are passed into the next module.

The **Output module** shows the query results, obtained in the previous module.

3 Conclusion and future work

The system we present is dedicated to XML structural retrieval without the need for the user to know XQuery and the XML documents precise structure. The system is based in a user-friendly QBE interface for specifying information needs. In a first step, the user visualises the set of existing XML Schema in the repository in the form of trees (alternatively graphs) and chooses the desired documents collection. After choosing the schema, a sample XML document from the corresponding collection is displayed. Having the schema tree view together with the sample document helps the user to better identify the document parts

Fig. 2. Selection mechanism

he needs. The user specifies the interesting elements or textual parts by just selecting them in the sample document view in an easy way.

As future work, we intend to perform some improvements, like a more powerful/intelligent selection algorithm to choose a better sample document.

References

1. D. Braga and A. Campi. Xqbe: A graphical environment to query xml data. *World Wide Web*, 8(3):287–316, 2005.
2. L. Kim. *The XMLSPY Handbook*. John Wiley & Sons, Inc., New York, NY, USA, 2002.
3. X. Lu. Document retrieval: A structural approach. *Inf. Process. Manage.*, 26(2):209–218, 1990.
4. S. Newman and Z. M. Ozsoyoglu. A tree-structured query interface for querying semi-structured data. *Scientific and Statistical Database Management, International Conference on*, 0:127, 2004.
5. R. Ramakrishnan and J. Gehrke. *Database Management Systems*, chapter 6. 2007.
6. J. H. G. Xiang Li and J. F. Brinkley. XGI: A Graphical Interface for XQuery Creation. In *American Medical Informatics Association Anual Symposium proceedings*, volume 2007, pages 453–457, November 2007.

An importer of virtual 3D City Models datasets into a spatiotemporal database

Wagner Franchin[1], Alexandre Carvalho[1], José Moreira[2], A. Augusto de Sousa[1] and Cristina Ribeiro[1]

[1] INESC Porto, Campus da FEUP, Rua Dr. Roberto Frias, 378, 4200-465 Porto, Portugal - [wjf, alexandre.carvalho, aasousa, mribeiro]@inescporto.pt
[2] IEETA, Campus Universitário de Santiago, Universidade de Aveiro, 3810-193 Aveiro, Portugal - jose.moreira@ua.pt

Abstract. Geographical Information Systems (GIS) have an important role in a wide range of application domains, both for the management of spatial data and as a decision support tool. For this reason virtual 3D city models are becoming increasingly complex with respect to their spatial, thematic structures and temporal aspects. In this paper we present Importer CityGML, a tool for parsing and importing XML datasets into a spatiotemporal database prototype system.

Keywords: Spatiotemporal Databases, CityGML, Virtual 3D City Models

1 Introduction

The need of 3D City Models is growing, especially for organizations involved in urban and landscape planning, cadastre, real estate, utility management, geology, tourism, army, etc. Geographical Information Systems (GIS) have an important role in a wide range of application domains, both for the management of spatial data and as a decision support tool. However, current GIS have been developed to deal efficiently with the current state of spatial data, but they do not include temporal features to deal with previous states about spatial information.

To cope with this limitation, Carvalho et al. [1] present a spatiotemporal database prototype system (*Action!*), developed on the top of an Oracle database management system and TimeDB [2], for efficient representation, querying and visualization of time-evolving urban information.

Further, CityGML [3] is a general information model for representing geovirtual 3D environments such as virtual 3D city models. It introduces classes and relations for topographic objects of urban environments and regional models.

This paper presents a tool to transfer information from a CityGML files into *Action!*, in order to benefit from the spatiotemporal querying and visualization capabilities of this system.

2 Related work

Generating virtual city environments has already been addressed in several works. The initial methods were based on computer aided architectural design where detailed measurement of the geometry was regarded as essential. Kolbe et al. [3] proposing a new model based on GML to describe data for any kind of city, Hagedorn and Dollner [4] describing an approach to visualize and analyze CAD-based 3D building information models (BIM) within 3D virtual city models, and Google SketchUp [5] allowing creating, visualizing and modifying 3D representations, are notable examples.

CityGML [3] is an open data model structure and standardized code based on XML for storing and exchanging virtual 3D city models. The common information model behind CityGML defines classes and relationships for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantic and appearance properties. The thematic model of CityGML considers thematic fields like Digital Terrain Models, sites (i.e. buildings), vegetation (solitary objects and also areal and volumetric biotopes), water bodies, transportation facilities or city furniture.

Finally [6] Kolbe et al. present a 3D geo database for CityGML. The CityGML data model is mapped into a relational database schema and an import/export tool was realized for processing of CityGML and GML structures. The system is not prepared to manage temporal information.

3 Importer CityGML

Importer CityGML is a tool for parsing and importing CityGML datasets into *Action!*. The system was developed in Java and uses classes and interfaces provided by XML Beans.

The **data model** defined in *Action!* is based on the CityGML schema and classes. As CityGML data files may hold objects with a null identifier, the primary key of each relation in *Action!* is a unique identifier generated automatically.

The CityGML class *Building* is mapped into a relation also named *Building*, to store the details about constructions, such as buildings and houses. The *OuterBuildingInstallation* relation stores information about the outer parts of buildings, for example, cash, chimneys or staircases. *BoundarySurface* relation holds information regarding the parts of a building, e.g. inner and outer walls, ceiling or roof.

The CityGML classes *CityFurniture*, *Generics*, *LandUse*, *Transportation*, *Vegetation* and *WaterBody* were aggregated into a single relation *ThematicObject*, as they share the same set of attributes. The *CityObjectGroup* class is represented in the database by the *CityObjectGroup* relation. This relation stores information about groups of objects in a city, for example, an university. *GroupMember* relation identifies the elements composing a group of objects, which can be groups of buildings or thematic objects.

The relation *Geometry Object* stores the geometric information and the level of detail of all city objects in CityGML data files. It also includes a temporal component defined by two additional attributes (start date and end date), denoting a valid time interval, and an attribute of type BLOB (Binary Large Object) to store all elements that make up a geometry in a JME (Java Monkey Engine [7]) compatible format, used by the *Action!* visualization tool.

SurfaceMember relation stores information about the type of geometry surfaces (Polygon, TexturedSurface, OrientableSurface or SurfaceMember) and *CoordinateGeometry* is the relation holding geometric coordinates of each surface. The coordinates are stored in an Oracle Spatial column of type Sdo_Geometry and so, they ready to be used by Oracle Spatial's operations.

To make the import into the database easier, each relation in the database model has a corresponding Java class. During the **parsing**, the information contained in the CityGML file is stored in a list of objects in Java. For instance, each building has a list of objects composed by geometry objects, boundary surfaces or outer building installations. Each boundary surface has another list of objects composed by geometry objects and so on.

After the parsing phase, the system has all file information in memory (in a city object list) and starts **importing data** into the database. The first relation to be filled is *CityModel*. After that, *Building*, *ThematicObject*, *CityObjectGroup* and *Appearance* are ready to be filled (see Fig. 1).

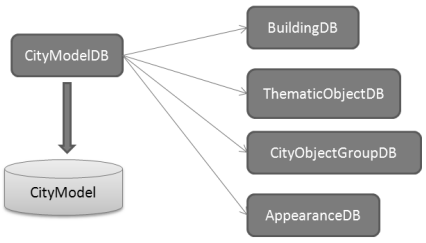
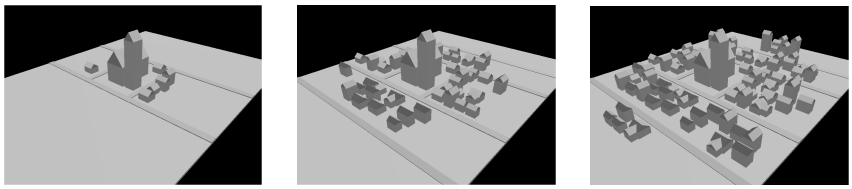


Fig. 1. CityModelDB class transfers data to CityModel table and passes the control to another class.

For example, when there is a Building in city object list, the system first imports the corresponding information into the *Building* relation and goes to the next element in this building object list. The next element can be a *GeometryObject*, *BoundarySurface* or *OuterBuildingInstallation*. If the system finds at this moment a *BoundarySurface* or an *OuterBuildingInstallation*, for example, the information is sent to the corresponding relations and the processing continues with the next object in the list.

Figure 2 illustrates a virtual 3D city at different dates, imported from CityGML into *Action!*. The original CityGML file was edited before importing to insert the temporal information.



(a) 1300 (b) 1700 (c) 2000

Fig. 2. Figures **(a)**, **(b)**, **(c)** shows the virtual in the year of 1300, 1700 and 2000, respectively.

4 Conclusion and future work

This paper presents the Importer CityGML, a tool for parsing and importing CityGML datasets into a spatiotemporal database system. The system has initially been realized as an Oracle 10G R2 Spatial relational database schema. After parsing the XML file, each CityGML object and its corresponding information are imported to the tables into the relational database. The main implementation goals are to achieve efficient storage, fast processing of CityGML and spatiotemporal management capability.

Future work will extend the system to parse and import other CityGML classes which have not been implemented, for example, ParameterizedTexture.

References

1. A. Carvalho, C. Ribeiro, and A. Sousa, "A Spatio-Temporal Database System Based on TimeDB and Oracle Spatial" in IFIP International Federation for Information Processing. Springer, 2006, pp. 205: 11-20.
2. Steiner, A., "A Generalization Approach to temporal Data Models and Their Implementations", Phd, Zurich, 1998.
3. T. Kolbe, G. Groger, and L. Plumer, "Interoperable Access to 3D City Models". in First International Symposium on Geo-Information for Disaster Management. Delft, Netherlands: Springer, March 21-23 2005.
4. B. Hagedorn and J. Dollner, "High-level web service for 3d building information visualization and analysis", in GIS 07: Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems. New York, NY, USA: ACM, 2007, pp. 1-8.
5. "Google Sketchup." [Online]. Available: http://sketchup.google.com/ (July 15th, 2009)
6. T. H. Kolbe, G. Knig, C. Nagel, and A. Stadler, "3d-geo-database for citygml - version 2.0.1", Institute for Geodesy and Geoinformation Science Technische Universitt Berlin, Tech. Rep., 2009.
7. "Java Monkey Engine." [Online]. Available: www.jmonkeyengine.com/ (July 15th, 2009)

Sessão 3B: Computação Distribuída e de Larga Escala

Análise do custo e da viabilidade de um sistema P2P com visibilidade completa

Simão Mata, J. Legatheaux Martins, Sérgio Duarte e Margarida Mamede

Departamento de Informática
Faculdade de Ciências e Tecnologia, FCT
Universidade Nova de Lisboa
2829-516 Caparica, Portugal,
simao.m@gmail.com, {jose.legatheaux, smd, mm}@di.fct.unl.pt,
http://di.fct.unl.pt

Resumo Este artigo apresenta um estudo da viabilidade de um algoritmo de filiação P2P com visibilidade completa, sendo definidos os limites para este tipo de aproximação através da análise do custo do algoritmo em função do dinamismo do sistema. É ainda proposta e avaliada a utilização de uma rede com super-nós, como forma de suportar cenários de maior dinamismo.

Abstract This paper presents a feasibility study for a P2P membership algorithm with full visibility. The limits for this kind of approach are evaluated in terms of the cost of the algorithm as a function of system churn. It is also proposed and evaluated the use of super nodes as the basis for supporting more dynamic settings.

1 Introdução

A difusão filtrada (*content-based networking*) é um problema que há largos anos interessa à comunidade científica [2] e ainda hoje é alvo de estudo. Esta forma de comunicação multi-ponto é uma generalização do paradigma editor-assinante baseado no conteúdo [1]. A particularidade deste modelo de difusão reside nos participantes poderem manifestar o interesse em receber apenas as mensagens cujo conteúdo respeita um dado padrão, o qual pode ser entendido como um *filtro*.

A difusão filtrada tem-se revelado um problema complexo e têm sido numerosas as soluções propostas ao longo dos anos [7]. De um modo geral, o problema tem sido atacado através da construção de árvores de difusão que reflectem, de alguma forma, o grau de afinidade dos filtros submetidos pelos participantes. Nesse processo procura-se minimizar a entrega de mensagens não desejadas (falsos positivos, ou simplesmente *spam*) e, ao mesmo tempo, evitar os *falsos negativos*, i.e., quando a difusão de uma mensagem falha alguns dos interessados.

O projecto LiveFeeds [1] está a desenvolver algoritmos P2P de difusão filtrada usando a disseminação cooperativa de *feeds RSS* como caso de estudo. Um dos

[1] Projecto PTDC/EIA/76114/2006, Project *LiveFeeds – P2P Dissemination of Web Syndication Content*, financiado pela FCT/MCTES.

objectivos deste projecto reside na optimização da difusão filtrada pela via de uma distribuição equitativa da carga pelos nós participantes.

LiveFeeds procura equacionar o problema da difusão filtrada como um problema de computação distribuída, em que a natureza interna dos filtros é relegada para segundo plano. O algoritmo LiveFeeds não produz falsos positivos, nem falsos negativos [2], entregando as mensagens a todos os nós interessados e apenas a esses. Além disso, o esforço despendido no encaminhamento de uma mensagem, que inclui a avaliação dos filtros, recai apenas nos nós a que ela se destina. Para tal, o algoritmo tem como requisito que cada nó participante tenha visibilidade completa e consistente da filiação do sistema. Esta característica enquadra o algoritmo LiveFeeds na filosofia *One Hop Routing* [5], exemplificado pelas DHTs que sacrificam a complexidade espacial das tabelas de encaminhamento para obter encaminhamento com custo constante. Trata-se de um requisito forte, porém a simplicidade e robustez que confere à solução para o problema da difusão filtrada justifica a abordagem que está a ser investigada.

O presente artigo pretende discutir a viabilidade da difusão filtrada LiveFeeds, através da análise do custo do algoritmo de filiação adoptado. Em concreto, pretende-se balizar o volume de tráfego investido pelos nós na manutenção de uma visão completa e consistente da filiação em função do dinamismo do sistema. Este será modelado em termos da taxa de entrada de novos nós e o seu tempo médio de vida no sistema.

Nas secções seguintes é descrito o algoritmo de filiação LiveFeeds e é feita uma análise teórica do mesmo. Segue-se uma discussão da validação e dos resultados obtidos por simulação. É, também, descrita e avaliada uma solução para o problema em que se traduz a entrada de um número elevado de participantes num curto espaço de tempo. O documento termina com as conclusões finais e uma breve descrição do trabalho que ainda resta desenvolver.

2 O algoritmo de filiação com visibilidade completa

O algoritmo utiliza a própria informação de filiação de cada nó para ir construindo árvores de difusão aleatórias com vista a disseminar rapidamente a entrada de novos nós pelo sistema. Um mecanismo epidémico complementar tem como função eliminar as inconsistências que inevitavelmente se produzem devido a entradas concorrentes e às ocasionais falhas dos nós.

Em concreto, cada nó LiveFeeds é conhecido por um identificador aleatório numérico (por hipótese, sem colisões e com uma distribuição uniforme), pelo seu endereço e por um filtro. O domínio dos identificadores está dividido num pequeno número de fatias (menos de uma dezena para cenários de utilização típicos). Em cada fatia, o nó com o identificador numericamente menor é, tendencialmente, conhecido como o líder da fatia (*slice leader*) e tem um papel destacado no algoritmo. Não existe qualquer tipo de coordenação entre os *slice leaders*. É ainda de notar que, momentaneamente, pode haver mais do que um

[2] Ignorando falhas nos nós, as quais exigem cuidados extra que não serão discutidos neste artigo.

slice leader em cada fatia, facto previsto pelo algoritmo. O sistema assume, também, que existe conectividade entre todos os nós.

Quando um nó pretende juntar-se ao sistema, fá-lo através de um nó já presente no sistema, conhecido a priori. Este nó auxiliar serve para encaminhar o pedido de entrada ao *slice leader* relevante, determinado com base no identificador do nó que pretende entrar. A função de cada *slice leader* consiste em agregar pedidos de entrada antes de iniciar a sua difusão pelos nós do sistema. Essa acumulação realiza-se até ser atingido um número razoável de eventos, ou até se esgotar o tempo máximo permitido de acumulação, fixado em cerca de 30 segundos. Nessa altura, é iniciada a difusão do evento de agregação pelo sistema através de uma árvore aleatória de grau G, gerada no decurso do processo de difusão. Para tal, o *slice leader* começa por dividir o universo dos identificadores em G sub-intervalos com a mesma cardinalidade (de nós). Para cada um deles, o *slice leader* selecciona um nó escolhido ao acaso, a quem entrega o evento de agregação a difundir e o sub-intervalo dos identificadores que ele, por sua vez, deverá tratar. O processo repete-se recursivamente, em cada nó da árvore aleatória assim gerada, até se chegar ao ponto em que os intervalos ficam tão estreitos que os nós que restam podem ser tratados simplesmente como folhas. Um nó sabe que entrou no sistema quando recebe a mensagem que difunde a sua entrada no sistema. Se tal não acontecer em tempo útil, ele repete todo o processo de entrada.

Sendo o processo de difusão anterior, essencialmente, *best-effort*, a visão da filiação do sistema que os nós assim adquirem é, inevitavelmente, imperfeita. Falhas ocasionais de nós e eventos de difusão concorrentes geram inconsistências que têm que ser corrigidas. Para tal, cada novo evento agregado de filiação, iniciado por um *slice leader*, é identificado por um número de sequência local único e uma estampilha (vectorial) global. Esta última tem como função registar o evento de filiação emitido mais recentemente por cada um dos *slice leaders* do sistema. Cada uma destas estampilhas globais representa uma "vista' concreta da composição do sistema, que cada nó pode comparar com a sua própria visão, computada a partir dos eventos de filiação que efectivamente recebeu, a fim de detectar eventos perdidos. A recuperação de eventuais omissões faz-se por epidemia. O processo é simples e consiste em periodicamente seleccionar um nó escolhido ao acaso e enviar-lhe a estampilha vectorial que identifica os eventos já vistos, na esperança de receber na volta os eventos em falta.

Note-se, ainda, que assim que um nó é tido (por qualquer outro) como tendo entrado no sistema, passa a poder se seleccionado para nó interior de uma árvore de difusão. Tal facto obriga que cada nó, previamente à sua entrada, obtenha uma cópia razoavelmente completa e actualizada da informação de filiação do sistema. Com o objectivo de reduzir a dimensão dessa informação e, de forma relacionada, reduzir a dimensão das estampilhas vectoriais usadas na recuperação epidémica, é imposto um limite máximo ao tempo de sessão dos nós. Porém, o impacto destas medidas e o seu custo ainda estão a ser avaliados e estão fora do âmbito deste documento, o qual se centra exclusivamente no custo do processo de difusão da informação de filiação em função do dinamismo do sistema ignorando o custo da reparação das falhas.

Sobre a visão completa e consistente, assim obtida, é implementado o processo de difusão filtrada. Essencialmente, trata-se de uma pesquisa distribuída dos nós cujos filtros aceitam a mensagem, ficando cada nó responsável por realizar uma parte do trabalho que resta, representado sob a forma de um intervalo de identificadores. Essa pesquisa faz-se através uma árvore aleatória que cobre apenas os nós cujo filtro aceita a mensagem em questão, ou seja apenas estes realizam esta computação distribuída.

A geração da árvore tem várias semelhanças com o processo de difusão da informação de filiação, sendo a principal diferença a parte que trata da selecção dos nós de cada nível inferior da árvore. Nomeadamente, geram-se também G' sub-intervalos. Porém, em vez de seleccionar um nó ao acaso para cada sub-intervalo obtido, esses sub-intervalos são processados sequencialmente até ser encontrado (em cada um deles) um nó, se houver, que aceite a mensagem. Os G' nós (no máximo) assim produzidos irão receber a mensagem a disseminar, mais o respectivo sub-intervalo que ainda restar. Para garantir a aleatoriedade da árvore, antes da proceder à divisão inicial, é primeiro adicionado um deslocamento aleatório.

Com este algoritmo não é produzido *spam*. A menos da ocorrência de falhas nos nós, também não há falsos negativos, pois quando um evento chega a um nó é possível atrasar o seu processamento até que esse nó chegue a uma vista de filiação igual ou posterior à estampilha de emissão da mensagem.

3 Análise do custo da difusão usando árvores aleatórias

Começaremos por analisar o custo da difusão global (*broadcasting*) num sistema estável (i.e, sem *churn*) com N nós. O objectivo é ter um ponto de partida que permita estimar os valores da capacidade de *upstream* e de *downstream* que cada nó terá de investir na implementação do algoritmo de filiação.

Por hipótese, vão ser difundidas continuamente mensagens, ao ritmo de r mensagens por segundo, por todos os N nós do sistema. Cada uma das mensagens tem a dimensão de m_t bytes. Pretende-se estimar a capacidade de *upstream* e de *downstream* que tal difusão requer. Quando um nó a pretende difundir uma mensagem m a todos os outros nós do sistema, selecciona um nó aleatoriamente, nó c, para ser a raiz da árvore de difusão. c dá então início à construção de uma árvore de difusão aleatória com grau G, processo que terá lugar simultaneamente com a própria difusão.

Para realizar esta análise, um primeiro passo consiste em determinar qual a probabilidade, $P(folha)$, de um nó ser folha e $1 - P(folha)$ de ser nó interior numa árvore de difusão. Para simplificar a análise, vamos admitir que as árvores usadas pelo algoritmo de difusão são equilibradas e todos os nós interiores têm grau G. Nestas condições, sendo h a altura da árvore de difusão formada, N satisfaz:

$$N = \sum_{i=0}^{h} G^i = \frac{G^{h+1} - 1}{G - 1} \tag{1}$$

$$P(folha) = \frac{G^h}{N} = \frac{G^h(G-1)}{G^{h+1}-1} = \frac{G^{h+1}-G^h}{G^{h+1}-1} \approx \frac{G^{h+1}-G^h}{G^{h+1}} = 1 - \frac{1}{G} \quad (2)$$

O tráfego de cada nó, aquando da recepção de uma mensagem de notificação, depende da posição do nó na árvore de difusão. As folhas da árvore só têm de receber a mensagem de notificação e não necessitam de a encaminhar para G nós, como acontece com os nós interiores. Um nó interior recebe m_t e envia $G.m_t$, enquanto uma folha não envia mensagens e recebe m_t.

A capacidade de *upstream* (b_u) e *downstream* (b_d) requerida em média por cada nó é dada por:

$$b_u = P(interior) \times G.m_t \times r = \frac{1}{G} \times G.m_t \times r = m_t \times r \quad (3)$$

$$b_d = m_t \times r \quad (4)$$

Podemos assim concluir que, em média, todos os nós considerados contribuem com o mesmo tráfego de *upstream* e *downstream*, pois a probabilidade de um nó ser folha é inversamente proporcional ao grau da árvore.

Dado que a análise anterior foi realizada com base numa restrição sobre os valores possíveis do número de nós do sistema (cf. a equação 1) e pressupondo que a árvore de difusão é completa e equilibrada, foram realizadas um conjunto de simulações que permitissem aferir se este resultado é uma boa aproximação dos requisitos do algoritmo de difusão no que diz respeito à capacidade usada na rede.

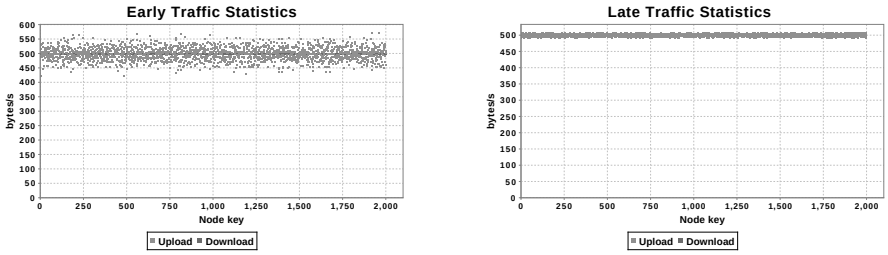
As simulações consistiram na criação de um sistema com N nós, com visibilidade completa e um tempo de sessão infinito, sobre o qual se desencadearam difusões totais, com raiz aleatória e à taxa de uma mensagem por segundo, de dimensão 500 Bytes. Os gráficos apresentados na figura 1 correspondem a uma dessas experiências, amostradas em dois momentos distintos. Esta e outras simulações, para vários valores de N, confirmaram que os valores médios do tráfego de *download* são os esperados e que os valores médios do tráfego de *upload* convergem para o mesmo valor com o número de difusões realizadas.

4 Custo da difusão da filiação num sistema dinâmico

Num sistema P2P real, os nós entram e saem do mesmo. Estes eventos devem repercutir-se em modificações da tabela de encaminhamento dos outros nós. No sistema em análise, os eventos que são relevantes são as entradas, na medida em que os eventos de saída não são propagados [3].

As entradas e saídas são eventos geralmente independentes e a repetição de eventos deste tipo cria um efeito que se designa por *churn* [11], o qual caracteriza

[3] Por hipótese, nesta fase do desenho, cada nó marca como inacessíveis os nós com que não consegue comunicar.



(a) (b)

Figura 1: Tráfego médio usado por um sistema de 2000 nós, no momento $t = 20$ segundos (a) e no momento $t = 3600$ segundos (b)

a dinâmica das alterações de filiação [3,9]. Um modelo de *churn* é composto por duas componentes. Uma distribuição que modela o tempo entre duas chegadas consecutivas de novos nós (*inter arrival time*) e uma distribuição que modela o tempo de sessão de cada nó [4]. Muitos sistemas P2P são avaliados com base em modelos simplistas, que usam distribuições uniformes do intervalo de chegada consecutiva de novos nós, e valores constantes do tempo de sessão. Isso pode ser enganador porque existem situações de *flash-crowds*, isto é, situações em que existe uma grande afluência de utilizadores num curto espaço de tempo (e que podem também ser caracterizadas por tempos de sessão muito curtos), correspondendo a um elevado *churn* que o sistema tem suportar. No contexto do LiveFeeds, uma *flash-crowd* pode corresponder a uma altura em que acontece algo importante e os utilizadores entram no sistema para procurar notícias e artigos sobre esse assunto, mas assim que obtêm o conteúdo pretendido abandonam o sistema.

Neste estudo procurou-se colmatar esse defeito de estudos anteriores (cf. [6,10]) usando um modelo mais realista. Em [4] é examinada a rede *skype* e é tanto quanto sabemos o único estudo que fornece alguma informação sobre o comportamento dos utilizadores nesta rede, que se pensa ter utilizadores que têm um comportamento semelhante aos possíveis utilizadores do *LiveFeeds*. Neste trabalho, é referido que na rede *skype* a entrada e saída de utilizadores depende fortemente da altura do dia e da altura da semana, havendo mais entradas durante a manhã, mais saídas ao fim do dia e mais utilização da rede em dias úteis. Esta ideia é também suportada por [12] que estuda o comportamento dos utilizadores de uma rede de *instant messaging* e por [8] que estuda o comportamento dos utilizadores de *e-mail*.

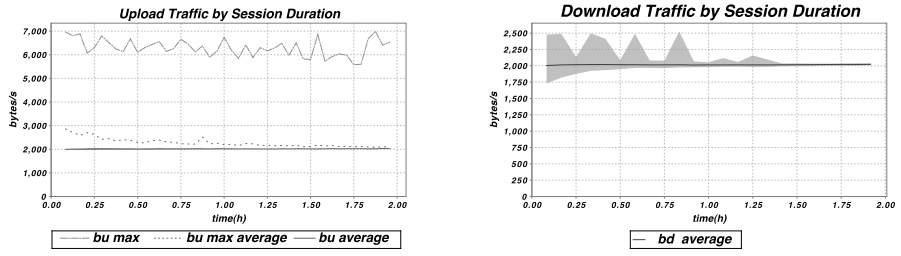
A partir destes estudos devemos modelar as chegadas de utilizadores através de um processo de *Poisson* não homogéneo, uma vez que a quantidade de entradas depende das alturas do dia e da semana, e modelar o tempo de sessão através de uma distribuição *heavy-tailed* [4], que de acordo com [12] pode ser uma distribuição tipo *Weibull*. Uma vez que apenas se pretende estudar o comportamento do algoritmo de filiação no pior caso possível, durante *flash-crowds*, é apenas necessário usar uma distribuição de *Poisson* homogénea que modele os valores de tempo entre chegadas consecutivas durante a pior altura.

O estudo dos requisitos do algoritmo de filiação para difundir as alterações da mesma, foi realizado através da simulação do algoritmo, fazendo vários testes, em que o sistema ia crescendo continuamente, com os parâmetros apresentados na tabela 1.

Tabela 1: Parâmetros usados na simulação

Parâmetro	Valor
G	4
Número de fatias	4
Periodicidade das difusões pelos SLs	25 a 30s ou 15 a 20 msgs
Tamanho de uma mensagem de filiação	500 Bytes
Tempo de sessão	300 a 7200 segundos
Distribuição de tempo de sessão	*Weibull* com $\lambda = 5000$ e $k = 0,5$
Distribuição dos tempo entre chegadas	*Poisson* com $\lambda = 1/2$, $1/4$ e $1/10$
Tempo de amostragem	25 segundos

Durante a simulação é registado o tráfego total enviado e recebido por cada nó. Estes dados permitem calcular a capacidade média (b_u *average* e b_d *average*) requerida por nó para execução do algoritmo. São também realizadas amostragens a cada 25 segundos, o que ainda permite determinar o valor máximo observado (b_u *max*) em cada período de amostragem e a média destes valores (b_u *max average*) para cada nó. A figura 2 apresenta os valores máximos observados (b_u *max*) no sistema e os médios das restantes grandezas acima descritas, para nós agrupados por tempo de vida.



(a) Tráfego de *upstream* (b) Tráfego de *downstream*

Figura 2: Resultados obtidos com distribuição do tempo entre chegadas parametrizada com $\lambda = 1/4$ (valor médio $r = 4$ *eventos*$/s$) e tempo de sessão máximo de 2 horas

Pela figura 2 observa-se que os dados obtidos para os valores médios estimados no fim de vida de cada nó são coerentes com os resultados teóricos descritos na secção 3 ($b_u = b_d = r \times m_t = 4,0 \times 500 = 2000$ B/s), apesar de não estarmos perante exactamente as mesmas condições.

Os valores obtidos para b_u *max* mostram o pior caso possível que os nós tiveram que suportar, uma vez que representa a capacidade máxima utilizada,

de entre todos os nós com o mesmo tempo de sessão. Esta métrica fornece assim um ponto extremo para o cálculo dos limites do algoritmo simulado. Neste caso, houve pelo menos um nó que necessitou de suportar o envio de $7000B/s$ num intervalo de $t_s = 25s$, tendo esse nó permanecido no sistema cerca de $1,9$ horas.

Em relação aos valores para a banda passante de *downstream* (Figura 2 (b)) não se nota uma variação significativa nos valores observados. Embora existam algumas flutuações, o valor médio situa-se sempre nos $b_d = r \times 500 = 2000B/s$, correspondente ao valor esperado no cenário ideal analizado na secção 3. Pela observação deste gráfico corrobora-se também o resultado já referido de que em média e ao fim de um número muito grande de *broadcasts* o valor da banda passante de *downstream* será igual ao valor da banda passante de *upstream*, uma vez que se verifica que $b_u = b_d \approx 2000B/s$. Tal facto é posto em evidência pelo facto de os valores mínimo e máximo de b_d *average* se aproximarem de b_d *average* com o aumento do tempo de sessão (c.f. área poligonal sombreada em (2)).

5 Introdução de super-nós

As análises anteriores mostram que os requisitos de comunicação do algoritmo são lineares com o ritmo médio de entrada de novos nós. Esta ideia foi em geral omitida nas anteriores análises de algoritmos com aproximação semelhante, c.f. por exemplo [6,10]. Tal custo não escala com aquele ritmo pelo que se torna necessário melhorar a escalabilidade do algoritmo.

Existem várias vias para melhorar este aspecto. No que se segue exploraremos uma que mantém a linearidade mas diminui a derivada e baseia-se na noção de "super-nó".

Este algoritmo suporta-se na ideia de que existem nós que são responsáveis pelos novos nós do sistema, os nós ditos *filhos* só se tornam super-nós e a sua entrada só é difundida para os todos os outros super-nós do sistema após algum tempo passado desde a sua entrada. Atenua-se assim o problema dos nós que estão muito pouco tempo dentro do sistema e provocam um aumento do *churn* observado, uma vez que cada entrada teria de ser difundida por todos os nós do sistema, mesmo que o nó que acaba de entrar tenha um tempo de sessão muito pequeno. Utilizando o novo algoritmo, a entrada de um novo nó só é difundida para todos os nós do sistema apenas quando ele se torna super-nó.

Introduz-se assim uma hierarquização do sistema, pois passam a existir nós de dois tipos distintos, os nós e os super-nós, sendo criada uma rede em que todos os super-nós conhecem os outros super-nós mas apenas conhecem os seus filhos.

No algoritmo reformulado, um nó a, para entrar no sistema, contacta um super-nó, s, enviando um pedido para que s seja pai de a. s deve decidir se pode aceitar mais um nó como filho ou reencaminhar a para outro nó. O processo repete-se até que algum nó aceite a como filho.

Após a ter encontrado um pai, deve aguardar uma mensagem de promoção vinda do pai, para que se junte à rede de super-nós. O filho fica assim a depen-

der do super-nó para receber mensagens referentes a acontecimentos na rede de super-nós.

Após algumas tentativas recusadas, o filho pode forçar a sua entrada. Ao receber um filho nestas condições, um super-nó deve promover um dos seus filhos para poder acomodar o novo nó. O critério que o super-nó utiliza para seleccionar o filho que deve promover tem grande influência no comportamento do sistema. Ao receber uma mensagem de promoção, o filho deve iniciar a rotina de entrada na rede de super-nós.

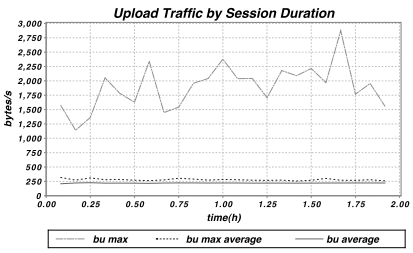
Como primeira aproximação, foram obtidas algumas simulações para que se pudesse determinar se uma aproximação deste tipo é viável. As simulações foram executadas utilizando os mesmos parâmetros utilizados durante as simulações referentes ao algoritmo normal (c.f. secção 4), bem como as mesmas métricas, de forma a que seja possível fazer uma comparação entre os dois algoritmos. Os resultados destas simulações são apresentados na figura 3. Uma vez que são medidos os valores considerados apenas para os super-nós, é considerado o tempo de sessão como super-nó e não o tempo de sessão total.

As simulações apresentadas na figura 3 correspondem a um caso em que ao entrar no sistema, um nó sabe sempre qual dos super-nós pode aceitar mais um filho e em que os super-nós sabem sempre qual dos seus filhos vai ficar mais tempo no sistema, promovendo esse nó sempre que seja necessário promover um dos seus filhos. Desta forma, o caso representado nesta simulação consiste no melhor caso possível.

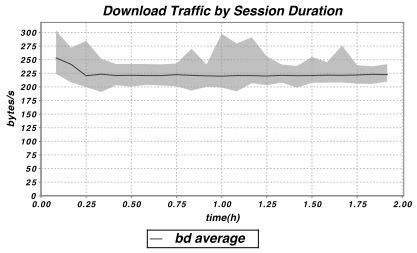
Como se pode notar pelo gráfico(a) da figura 3, o valor do tráfego de *upstream* utilizado pelos nós desceu significativamente, pois passa a depender linearmente do número de entradas por segundo na rede de super-nós e não do número de entradas por segundo em todo o sistema. A mesma descida é verificada ao observar o gráfico (b). É importante verificar que a igualdade entre o tráfego de *upstream* e *downstream* continua a ser observável, em média e ao fim de algum tempo, a quantidade de informação recebida é igual à quantidade de informação enviada.

Podemos assim concluir que a introdução de uma hierarquia deste tipo no sistema contribuiu de forma positiva para a diminuição do *churn* observado pela rede de super-nós e que esta aproximação tem assim um grande potencial de melhoramento.

Não havendo um oráculo perfeito para encontrar um super-nó disponível e para seleccionar o melhor filho a promover, como discutido anteriormente, a figura 4 apresenta os resultados obtidos para condições mais realistas. Nestas simulações, tanto a selecção do super-nó semente como dos filhos a promover são feitas através de um processo estocástico. Como seria expectável, comparativamente ao cenário correspondente à figura 3, observa-se um aumento da capacidade utilizada. Porém, pode-se verificar que continua a existir um ganho no que se refere ao tráfego de *upstream* e *downstream* que cada nó despendeu para difundir a informação de filiação dos novos super-nós, face ao algoritmo de visibilidade completa.

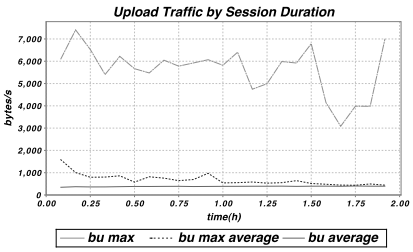


(a) Tráfego de *upstream*

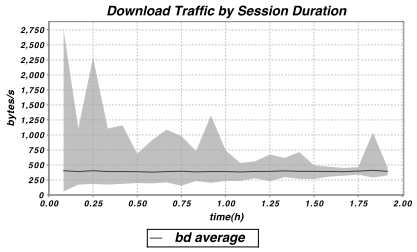


(b) Tráfego de *downstream*

Figura 3: Resultados obtidos utilizando o algoritmo de super-nós configurado de forma equivalente à dos ensaios da secção 4 perante condições óptimas de selecção de nós semente e nós a promover



(a) Tráfego de *upstream*



(b) Tráfego de *downstream*

Figura 4: Resultados obtidos utilizando escolha aleatória de nós semente e selecção aleatória de filhos a promover

Outros melhoramentos possíveis

Um dos melhoramentos a introduzir seria a implementação de novos critérios para a selecção do filho a promover quando é necessário introduzir um novo nó na rede de super-nós. Estes critérios podem tomar um papel de grande relevo no melhoramento do algoritmo. Um critério possível seria dividir os nós em classes, baseados em informação como a utilização de *NAT* por parte do nó, qual a capacidade estimada do nó ou outros factores que possam ser determinantes e seleccionar o nó mais apto para ser promovido para super-nó.

A noção de classes pode ainda ser desenvolvida para um outro modelo em que cada nó utiliza um critério de ordenação (*ranking*) para decidir qual dos seus filhos deve promover a super-nó. A posição de um nó poderia ser determinada em função da classe do nó e, também, através de conhecimento de sessões passadas do nó em questão. Nestes moldes, um nó precisaria de guardar de forma persistente vária informação sobre os nós, nomeadamente informação sobre duração de sessões passadas.

A introdução de novos nós na rede de super-nós pode ser feita de forma adaptativa de maneira a que os super-nós possam ter algum controlo sobre o número de entradas por segundo observadas. Podemos partir da ideia de que um super-nó tem conhecimento total dos eventos dentro da rede de super-nós e pode assim decidir se o número de entradas observadas no passado próximo é

baixo o suficiente para que a rede suporte uma nova entrada sem que exista uma sobrecarga. É preciso no entanto ter em consideração medidas para que o *feedback* proveniente deste tipo de mecanismo seja evitado. Se não forem implementadas tais medidas, vários nós podem detectar um baixo número de eventos de entrada e promover simultaneamente vários nós, levando a um aumento excessivo do número de entradas por segundo.

Ao ter de gerir filhos, um super-nó terá de utilizar mais capacidade de *upstream*, mas essa desvantagem poderá ser anulada pela redução do número de entradas observadas que levam a uma diminuição da capacidade necessária. Para reduzir mais a desvantagem que um super-nó tem ao ser responsável por outros nós, pode ser desenvolvido um mecanismo em que os filhos fazem o envio de mensagens no lugar do super-nó, participando na árvore de difusão no lugar do seu pai. O trabalho do super-nó pode ainda ser diminuído se os filhos difundirem entre si as mensagens vindas do super-nó.

6 Conclusões e trabalho futuro

Com base na análise feita, podemos tirar algumas conclusões sobre a aplicabilidade do algoritmo de filiação LiveFeeds com visibilidade completa. A utilização de árvores de difusão aleatórias tem como benefício que a capacidade de *upstream* necessária é proporcional ao número de eventos por segundo observada no sistema. A capacidade que cada nó deve suportar para que o mecanismo considerado possa ser utilizado não depende assim do tamanho do sistema, mas sim do comportamento dos seus utilizadores. Se esse comportamento for estável o suficiente, o mecanismo de visibilidade completa pode ser utilizado com um grande número de nós.

Neste sentido, podemos considerar o uso do mecanismo de visibilidade completa num sistema em que os nós comportam-se como *brokers* e poderão estar alojados em instituições de ensino, *ISPs*, grandes empresas, etc. e que servem os *feeds* aos utilizadores. Este tipo de nós têm tipicamente boa conectividade e tempos de sessão longos, o que leva a que, num sistema constituído maioritariamente por este tipo de nós, a taxa de eventos por segundo seja baixa o suficiente para viabilizar que seja suportado um grande número de nós. A título de exemplo, se considerarmos uma rede com $N = 10000$ nós, em que os nós têm um tempo sessão médio de cerca de 3 horas, isto equivale a uma taxa de eventos de cerca de $10000/10800 \approx 0,9$ eventos/segundo. Os nós teriam assim de contribuir com uma capacidade de *downstream* e *upstream* de $r \times m = 0,9 \times 500 = 450$ bytes por segundo para manter a informação de filiação, sendo este um custo bastante baixo para nós do tipo considerado.

Porém, para que o sistema suporte utilizadores com um comportamento mais dinâmico, é necessário encontrar outras soluções, como a introdução de super-nós. Através de super-nós afigura-se possível resolver dois problemas complicados e ignorados em muitos sistemas P2P. São eles, o elevado custo dos nós com tempos de sessão muito curtos e a entrada de muitos nós num curto espaço de tempo.

Em contrapartida, ao limitar a visão completa do sistema aos super-nós perde-se alguma da simplicidade do algoritmo de difusão filtrada inicialmente pensado no projecto LiveFeeds. Para evitar falsos positivos torna-se necessário que cada nó, até ser promovido, esteja sob a alçada de um super-nó cujo filtro seja pelo menos igual ao seu, ou em alternativa mais abrangente. Dado o conhecimento alargado que todos os super-nós possuem, esse processo terá custos de comunicação mínimos, mas obriga que o algoritmo de filiação passe a ter algum conhecimento da composição interna dos filtros.

Em termos de trabalho futuro, há ainda alguns estudos a realizar. Além de uma análise cuidada dos outros melhoramentos propostos na sub-secção 5, é preciso estudar as questões relacionadas com a dimensão da informação de filiação que cada nó precisa de receber antes de entrar no sistema. Caso o problema se revele excessivamente sério, tudo indica que uma forma de minimizar essa informação passará por dar ainda mais relevância aos super-nós, aumentado o número dos nós que deles dependem. E, como também já foi sugerido, entregando a esses nós alguma da responsabilidades de encaminhamento das mensagens de filiação.

Referências

1. A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *INFOCOM*, 2004.
2. P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.
3. P. B. Godfrey, S. Shenker, and I. Stoica. Minimizing churn in distributed systems. *SIGCOMM Comput. Commun. Rev.*, 36(4):147–158, 2006.
4. S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proceedings of The 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, pages 1–6, 2006.
5. A. Gupta, B. Liskov, and R. Rodrigues. One hop lookups for peer-to-peer overlays. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, pages 7–12, Lihue, Hawaii, May 2003.
6. A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. *Proc. First Symposium on Networked Systems Design and Implementation*, 2004.
7. J. Legatheaux Martins and S. Duarte. Routing Algorithms for Content-based Publish/Subscribe Systems. *IEEE Communications Tutorials and Surveys – Accepted for Publication*, page 21, 2009.
8. R. D. Malmgren, D. B. Stouffer, A. E. Motter, and L. A. N. Amaral. A Poissonian explanation for heavy tails in e-mail communication. *Proceedings of the National Academy of Sciences*, 105(47):18153–18158, 2008.
9. S. C. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. *Handling Churn in a DHT*. Computer Science Division, University of California, 2003.
10. R. Rodrigues and C. Blake. When Multi-Hop Peer-to-Peer Lookup Matters. In *In Proc. of IPTPS*, pages 112–122, 2004.
11. D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202, New York, NY, USA, 2006. ACM.
12. Z. Xiao, L. Guo, and J. Tracey. Understanding instant messaging traffic characteristics. In *Proceedings of the 27th International Conference on Distributed Computing Systems*, page 51. IEEE Computer Society Washington, DC, USA, 2007.

Custo da Comutação Dinâmica de Protocolos de Comunicação

Cristina Fonseca, Liliana Rosa, and Luís Rodrigues

IST/INESC-ID,{cfonseca,lrosa}@gsd.inesc-id.pt,ler@ist.utl.pt

Resumo Este artigo foca-se na adaptação dinâmica dos protocolos de comunicação que suportam as aplicações distribuídas. Em particular, o artigo apresenta duas contribuições: i) dois protocolos de comutação específicos e ii) avaliação quantitativa dos ganhos que se podem obter ao usar comutadores específicos em detrimento de um comutador genérico, usando como caso de estudo o seguinte conjunto de serviços: serviço de ordenação FIFO, serviço de ordenação total de mensagens em difusão e serviço de ordenação causal de mensagens. A análise destes três casos particulares oferece uma melhor compreensão das relações custo-benefício entre estes três tipos de comutadores.

Abstract This paper addresses the dynamic adaptation of communication protocols. Namely, it provides the following contributions: i) it proposes two novel protocol switching algorithms and, ii) it provides a quantitative assessment of the benefits of using failure switching algorithms against the use of generic switching algorithms. The following communication protocols are used as case studies: totally ordered multicast, causally ordered multicast and FIFO reliable multicast. The experimental results offer a better insight on the tradeoffs involved in the execution of protocol switching algorithms.

1 Introdução

Durante a execução de aplicações distribuídas ocorrem frequentemente mudanças de contexto, tais como alterações à capacidade de processamento disponível, na latência da rede, etc. Uma das formas de optimizar o desempenho destes sistemas consiste em adaptar dinamicamente o seu comportamento em resposta às alterações de contexto. Este artigo foca-se na adaptação dinâmica dos protocolos de comunicação que suportam a aplicação. Esta adaptação pode ser conseguida comutando, em tempo de execução, um protocolo por outro que ofereça o mesmo serviço.

A comutação dinâmica é orquestrada por um protocolo designado por *comutador*. Dois tipos de comutadores tem sido propostos na literatura: i) comutadores *genéricos* [1], que podem ser aplicados a um leque vasto de protocolos de comunicação que oferecem diferentes serviços, desde que estes protocolos possuam um conjunto mínimo de propriedades em comum; ii) comutadores *especializados* [2, 3, 4] que apenas podem ser aplicados para trocar concretizações alternativas de um único serviço em particular.

Os comutadores genéricos permitem reduzir o número de comutadores que é necessário suportar no sistema e os comutadores específicos podem optimizar o seu desempenho tirando partido de propriedades específicas dos serviços para os quais se destinam. Este artigo faz uma avaliação quantitativa dos ganhos que se podem obter ao usar comutadores específicos em detrimento de um comutador genérico para três serviços concretos: serviço de ordenação total de mensagens em difusão, serviço de ordenação causal e serviço de ordenação FIFO.

O resto do artigo está organizado do seguinte modo. A Secção 2 descreve o trabalho relacionado. Na Secção 3 descrevem-se os protocolos que são alvo do nosso estudo. Na Secção 4 faz-se uma análise comparativa do desempenho do comutador genérico e dos comutadores específicos. Finalmente, a Secção 5 conclui o artigo.

2 Trabalho Relacionado

A comutação dinâmica entre protocolos de comunicação é uma área onde já existe bastante trabalho realizado, tanto a nível da especificação dos protocolos de comutação como no desenvolvimento de arquitecturas de software que facilitam a composição de protocolos e a reconfiguração destas composições em tempo de execução.

No que se refere ao nosso alvo de estudo, os protocolos de comutação, a aproximação mais comum para comutar entre A e B consiste em utilizar a estratégia que designaremos por *comutação por paragem* [5]. Esta estratégia deixa o serviço indisponível durante um período de tempo que pode ser significativo. Desta forma, têm sido propostas estratégias alternativas que tentam minimizar o período de indisponibilidade do serviço durante a comutação. É neste contexto que surgem as soluções genéricas e especializadas que referimos anteriormente.

Os comutadores genéricos [1] são concebidos para satisfazer um conjunto de propriedades e a capacidade que um protocolo tem de ser aplicável a vários casos advém das hipóteses que este faz acerca dos protocolos alvo assim como das propriedades que o comutador preserva durante a comutação.

Os comutadores especializados [2, 6, 3] usam as propriedades específicas dos protocolos de comunicação para os quais foram desenhados para minimizar a interferência no fluxo de pedidos durante a comutação.

3 Casos de Estudo

Para analisar tanto os comutadores genéricos bem como os comutadores especializados, vamos comparar o desempenho de um comutador genérico (que designaremos por G) com três comutadores especializados, um concebido para comutar entre protocolos de ordem total (que designaremos por comutador T), outro concebido para comutar entre protocolos de ordem causal (que designaremos por comutador C) e um último que permite comutar entre protocolos de difusão melhor-esforço com ordem FIFO (será designado por F). Os testes usarão

composições de protocolos semelhantes à ilustrada na Figura 1. A Figura mostra ainda a interacção do coordenador da adaptação (cujo papel iremos descrever mais à frente) com os nós do grupo de comunicação.

No topo da pilha encontra-se a aplicação, que é responsável por invocar os serviços de comunicação fornecidos pela composição de protocolos subjacente. Mais abaixo, encontra-se a camada *comutador*, seguindo-se A e B que são diferentes protocolos que oferecem o mesmo serviço. O objectivo da arquitectura é permitir comutar da utilização de A para a utilização de B.

Desta composição, as camadas essenciais para a comutação são o comutador e o multiplexer. O comutador, que em cada teste poderá ser instanciado por um comutador G, T, C ou F, é responsável por armazenar estado, nalguns casos mensagens e ainda por enviar mensagens de controlo específicas do protocolo de comutação. É esta camada que efectua todo o processamento relativo à comutação entre os protocolos A e B da camada abaixo. O multiplexer é um componente cujo papel é esconder a existência de duas concretizações diferentes do serviço acima (A e B) das camadas inferiores da composição.

Em termos de canais, como ilustra a Figura 1, existe um canal u que liga a aplicação ao comutador. Este é o canal de comunicação que é utilizado pela a aplicação para trocar mensagens em difusão. O comutador possui dois canais, a e b, para comunicar com a camada multiplexer. Um dos canais utiliza o protocolo A e o outro o protocolo B. Em regime estável, todas as mensagens recebidas pelo canal u são encaminhadas apenas para a (ou b). Durante o processo de comutação, aplicam-se os algoritmos referidos anteriormente. O multiplexer despacha as mensagens recebidas por a e b para um único canal g que concretiza a comunicação em grupo fiável. Mais concretamente, o canal g usa uma composição de protocolos, que faz parte da distribuição do Appia, e que assegura propriedades de *sincronia na vista*[7].

3.1 Coordenação da Comutação

Assumimos que a comutação entre o protocolo A e B (e vice-versa) é activada por uma entidade logicamente centralizada designada por *coordenador*. O coordenador é responsável por comunicar com o comutador existente em cada nó para dar início ao processo de comutação. Neste trabalho, as estruturas de suporte à comutação estão sempre presentes na composição de protocolos pois o protocolo genérico que usámos obriga a que cada comutador informe os restantes do número de mensagens enviadas e consequentemente que as conte.

3.2 O Comutador G

Como comutador genérico escolhemos o comutador descrito em [1]. O protocolo concretizado resume-se no seguinte: quando o coordenador informa o comutador que é necessário comutar entre A e B, G envia uma mensagem a todos os participantes informando-os do número de mensagens que até agora tinham sido enviadas através de A. Esta é a última mensagem enviada através de A e a

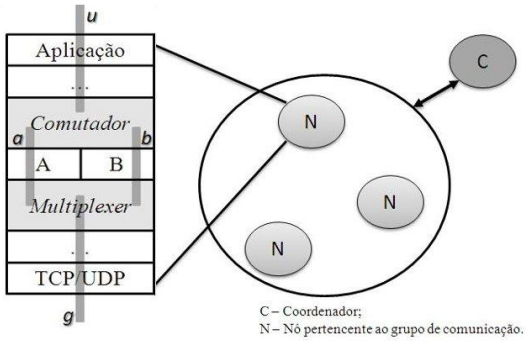


Figura 1. Composição com comutador.

partir daqui todas as mensagens enviadas passam a ser processadas através de B. Ao receber esta mensagem de todos os outros nós, cada comutador sabe quantas mensagens deveria ter recebido de cada um dos outros nós, através do protocolo A. G espera até ter recebido todas as mensagens de todos os nós para poder desactivar A. Finalmente, o comutador informa o coordenador que a comutação foi concluída.

Neste comutador, a necessidade de esperar por, e entregar, todas as mensagens enviadas pelo protocolo A antes de entregar mensagens enviadas pelo protocolo B é fundamental para que não sejam violadas as propriedades do protocolo subjacente, pelo que este comutador é susceptível a suspender as entregas se uma das mensagens enviadas em A sofrer atrasos.

3.3 O Comutador T

Informalmente, um protocolo de ordem total para comunicação em grupo garante que quaisquer duas mensagens entregues a quaisquer dois processos, são entregues na mesma ordem a cada um destes processos. Como já referimos, o comutador que designamos por T permite comutar entre dois protocolos de ordem total. Neste trabalho usamos o comutador T proposto por Mocito et al.[2] que, de um modo geral, funciona da seguinte forma. Durante a fase de comutação os eventos são processados e enviados simultaneamente pelos dois protocolos, sendo que a primeira mensagem enviada também por B é marcada. Na recepção, todas as mensagens enviadas através de A são entregues à aplicação e as que chegam por B são guardadas temporariamente. Depois de um nó ter recebido a primeira mensagem marcada de todos os outros nós, pode entregar à aplicação as mensagens guardadas que ainda não tinham sido entregues e descartar todas as outras. A partir deste momento o protocolo A deixa de processar mensagens.

Note-se que o comutador T pressupõe que o protocolo subjacente entrega as mensagens de acordo com a ordem total, pelo que poderá não funcionar noutros cenários. Dos comutadores aqui apresentados, apenas este obriga a enviar mensagens simultaneamente em dois protocolos. Isto inclui uma sobrecarga adicional que é discutida em pormenor em [2].

3.4 O Comutador C

Um protocolo de ordenação causal, assegura que as mensagens são entregues de acordo com as relações potenciais de causa-efeito, tal como descrito em [8]. Neste protocolo, duas mensagens que estejam potencialmente relacionadas de forma causal são entregues pela mesma ordem a todos os participantes; pelo contrário, mensagens concorrentes podem ser entregues por ordem distinta a participantes distintos. A função de um comutador para protocolos de ordem causal é assegurar que se preservam as relações de ordem entre as mensagens enviadas pelo protocolo A e as mensagens enviadas pelo protocolo B.

Não tendo encontrado na literatura nenhum comutador especializado para ordem causal, desenvolvemos o comutador C como uma adaptação do protocolo genérico G [1]. Tal como o comutador G, o comutador C também requer a contagem explicita de mensagens. No entanto, a entrega das mensagens enviadas por um nó não fica dependente do momento em que os restantes nós aplicam a comutação. O comutador C proposto funciona da seguinte forma. O comutador concretiza um protocolo de ordenação causal, baseado em relógios vectoriais[9], que apenas ficam activos durante a comutação. Em regime estável, antes da comutação, o comutador limita-se a atribuir um número de sequência local a cada mensagem que envia, e a registar o número de sequência das mensagens que entregou de cada um dos restantes processos.

Um comutador, ao iniciar a comutação, deixa de enviar mensagens pelo protocolo A e passa a enviar as mensagens pelo protocolo B. No entanto, acrescenta também um relógio vectorial às mensagens enviadas por B. Este relógio deixará de ser enviado quando a comutação terminar, mas permite ordenar mensagens enviadas por A em relação a mensagens enviadas por B durante a comutação. Neste algoritmo, um comutador inicia a comutação quando o primeiro dos seguintes eventos ocorrer: i) o comutador recebe uma instrução para comutar, vinda do coordenador ou; ii) o comutador recebe uma mensagem pelo protocolo B (o que significa que é necessário ordenar mensagens recebidas através de A e de B). Durante o processo de comutação, as mensagens recebidas pelo protocolo A são entregues sem restrições e as mensagens recebidas pelo protocolo B são entregues de acordo com o seu relógio vectorial. A comutação termina assim que todos os participantes estiverem a usar o protocolo B. Nesse momento, a utilização do relógio vectorial pelo comutador deixa de ser necessária, bastando garantir as seguintes restrições: i) a ordem FIFO entre as mensagens enviadas com e sem relógio vectorial é respeitada e; ii) as mensagens são entregues pela ordem que são recebidas do protocolo B.

3.5 O Comutador F

Um protocolo de ordenação FIFO assegura que as mensagens são entregues aos destinatários na ordem pela qual foram enviadas pelo remetente. Note-se que, ao contrário da ordem total, a ordem FIFO só reordena a entrega de mensagens que tenham sido enviadas pelo mesmo remetente. A função de um comutador

para protocolos de ordem FIFO é assegurar que se preserva a ordem FIFO entre a última mensagem enviada por A e a primeira enviada por B.

Não tendo encontrado na literatura nenhum comutador especializado para ordem FIFO, desenvolvemos o comutador F como uma adaptação do protocolo genérico G [1]. A nossa adaptação evita a contagem explicita de mensagens pelo comutador, assim como a sincronização entre os diversos canais FIFO. O comutador F proposto funciona da seguinte forma: após o comutador receber a informação de que deverá iniciar o processo de comutação, deixa de enviar as mensagens através do protocolo A e passa a enviá-las por B. Para assinalar a transição, a última mensagem enviada pelo protocolo A é marcada. No lado do receptor, mensagens recebidas por B vindas de um dado emissor são armazenadas até que a última mensagem enviada por este emissor através do protocolo A seja entregue. Nesse momento, são entregues todas as mensagens recebidas por B, pela ordem em que foram recebidas.

4 Avaliação

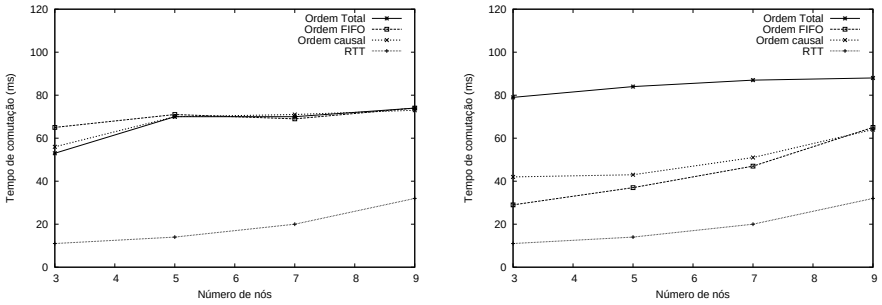
Nesta secção é avaliado o desempenho dos protocolos de comutação genéricos (comutador G) face aos protocolos específicos (comutadores T, C e F).

4.1 O Ambiente de Execução

A comutação dinâmica de protocolos é bastante facilitada se existir uma arquitectura de software de apoio à composição e execução de protocolos de forma eficiente e flexível. Encontram-se na literatura diversos sistemas de suporte à composição de protocolos que satisfazem estes requisitos, tais como o Cactus [10] ou o Appia [11]. Para suportar o nosso trabalho escolhemos o sistema Appia. Os testes foram efectuados em ambiente real numa grid constituída por 17 Intel Pentium IV (1-core) @ 3.2GHz / 1G RAM e 14 Intel Q6600 (4-core) @ 2.4GHz / 8G RAM organizados numa subrede gigabit privada. Desta rede faz também parte 1 AMDOpteron (dual) @ 2.4GHz / 16G RAM numa subrede gigabit pública. A arquitectura é a apresentada anteriormente, o código desenvolvido em Java e executado no nível de utilizador.

4.2 Métricas

A avaliação incide sobre duas métricas de desempenho: o tempo total necessário para realizar a comutação e o impacto que a comutação tem sobre a taxa de entrega das mensagens à aplicação. O tempo para realizar a comutação é o tempo medido entre o coordenador enviar a primeira mensagem com a instrução de iniciar a reconfiguração para um comutador e receber a última confirmação de que a reconfiguração foi concluída. Para perceber o peso computacional do algoritmo mediu-se também o tempo de comunicação entre o coordenador e os comutadores num sistema em que o comutador responde de imediato (RTT), que se indica nos gráficos como forma de comparação. Para avaliar estas métricas, testámos vários cenários cujos resultados se descrevem de seguida.



(a) Comutador genérico (b) Comutador específico

Figura 2. Efeito de variação do número de nós

4.3 Variação do Número de Nós

Esta medida pretende avaliar o comportamento dos diferentes comutadores, em função do número de nós envolvidos na comunicação. Esta avaliação foi feita para 3, 5 ,7 e 9 nós, mantendo-se a taxa de envio de mensagens constante e igual a 100msgs/s.

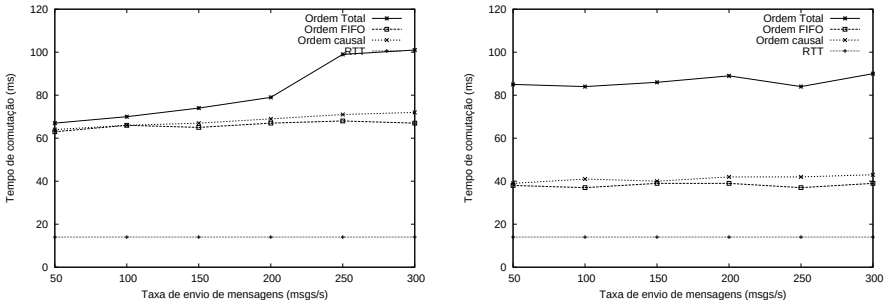
Como se observa na Figura 2, em ambos os casos (comutador genérico e comutador especializado), o tempo total necessário para comutar entre os protocolos varia de forma aproximadamente constante, aumentando ligeiramente com o aumento do número de nós. Sendo este aumento bastante idêntico nos dois casos de comutadores, podemos dizer que o protocolo não tem qualquer influência sobre esta variável. A variação do número de nós interfere assim ligeiramente no tempo de troca pois é necessário tempo adicional para difundir a mensagem com a ordem para comutar (mais nós aumentam o número de mensagens na rede e o processamento efectuado, o que também se traduz no aumento dos valores de RTT).

Podemos ainda concluir através da análise da Figura 2(a), que no caso do comutador genérico todos os protocolos têm tempos de comutação semelhantes. No caso dos comutadores específicos (Figura 2(b)), o tempo necessário para comutar nas diversas situações está relacionado com os requisitos que a ordem que se quer preservar impõe. Em termos de garantias, o protocolo mais exigente é o de ordem total, seguindo-se o de ordem causal e por fim o de ordem FIFO, ou seja, a ordem de complexidade é também a ordem a nível de tempo necessário.

4.4 Variação da Taxa de Envio de Mensagens

Com o objectivo de avaliar o comportamento do comutador entre protocolos para diferentes taxas de envio de mensagens, fez-se variar esta taxa em 50, 100, 150, 200, 250 e 300 msgs/s, mantendo-se o número de nós constante e igual a 5. Mais uma vez a métrica usada foi o tempo total de troca.

Analisando a Figura 3(a) concluímos que, para os comutadores genéricos, o tempo de comutação se mantém aproximadamente constante com excepção do



(a) Comutador genérico (b) Comutador específico

Figura 3. Efeito de variação da taxa

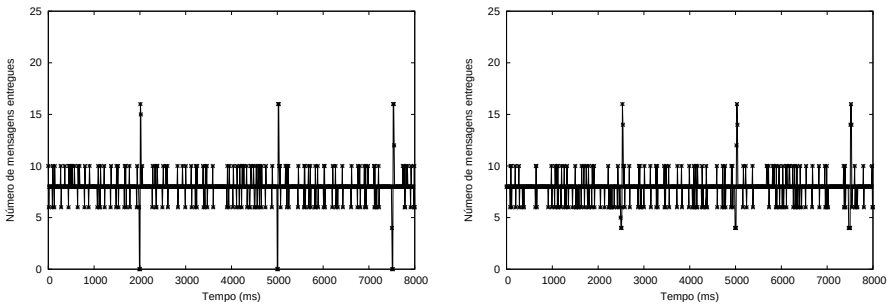
comutador genérico de ordem total em que há um aumento do tempo necessário para comutar, com o aumento do número de mensagens para processar. Este aumento é visível a partir de um certo valor de taxa (entre 200 e 250 msgs/s), após o qual se mantém constante. No que diz respeito aos comutadores específicos (Figura 3(b)), as variações observadas no tempo total de comutação em função do aumento da taxa de envio de mensagens são mínimas (mesmo para taxas superiores a 200 msgs/s). Nos casos dos comutadores de ordem causal e FIFO, os tempos de comutação são menores.

Se compararmos os dois comutadores de ordem total, concluímos que a variação que se observa no caso genérico se deve ao armazenamento temporário de mensagens. Considerando o seu funcionamento, para o caso do comutador específico (em que são sempre armazenadas mensagens durante o período de comutação), esta variação da taxa não se verifica e o tempo de comutação apresenta pequenas variações com a variação da taxa de envio de mensagens. No caso do comutador genérico este protocolo armazena mensagens a partir de uma certa taxa, sendo necessário tempo adicional para se efectuar a comutação. O tempo adicional é o necessário para reservar, manipular e libertar os recursos associados a esse armazenamento.

4.5 Taxa de Entrega das Mensagens à Aplicação

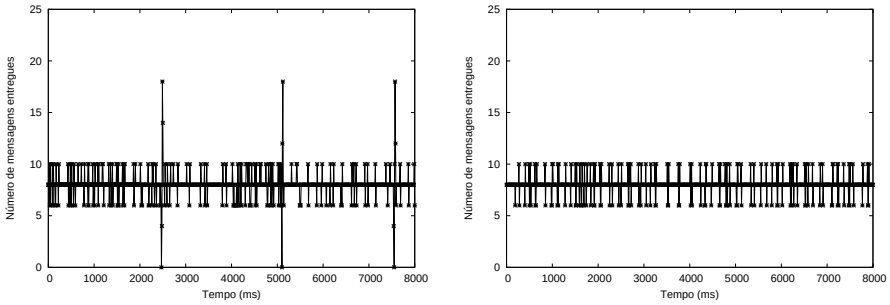
Outra forma de avaliar e comparar os diferentes protocolos de comutação, no que diz respeito ao impacto que estas operações têm para a aplicação, é estudar o número de mensagens que lhe são entregues e analisar a variação durante o período de comutação. Neste caso usou-se como métrica o número de mensagens entregues à aplicação, a cada 10 ms, durante 3 execuções do protocolo consecutivas,para um nó pertencente a um grupo de comunicação constituído por 5 elementos.

Por análise da Figura 4, concluiu-se que, para os protocolos de ordem total, enquanto o genérico inibe a entrega de mensagens à aplicação, o específico apenas diminui a taxa de entrega durante o período de comutação. Esta inibição ou não



(a) Genérico (b) Específico

Figura 4. Entrega das mensagens à aplicação - ordem total.



(a) Genérico (b) Específico

Figura 5. Entrega das mensagens à aplicação - ordem causal.

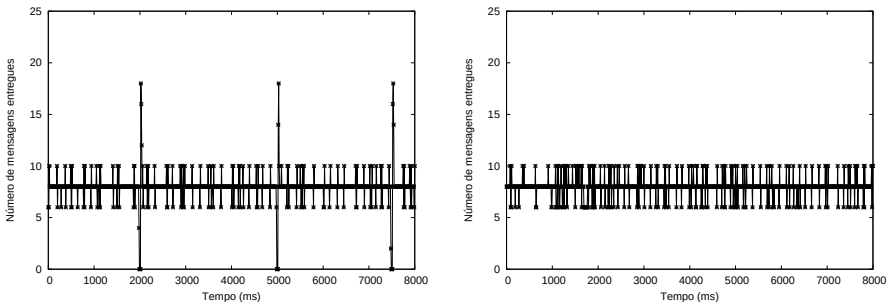
de mensagens não está relacionada com a especificidade do protocolo mas sim com a abordagem escolhida para o concretizar.

Analisando as Figuras 5 e 6 relativas aos protocolos de ordem causal e FIFO, para o caso genérico há interrupção na entrega de mensagens (de forma semelhante ao que acontece no comutador genérico para ordem total), também devido ao armazenamento temporário de mensagens, enquanto no caso específico essa interrupção não ocorre.

Conclui-se que, dependendo dos casos, é possível realizar alterações na composição de protocolos que suportam a comunicação, em tempo de execução, minimizando a interferência para as camadas superiores.

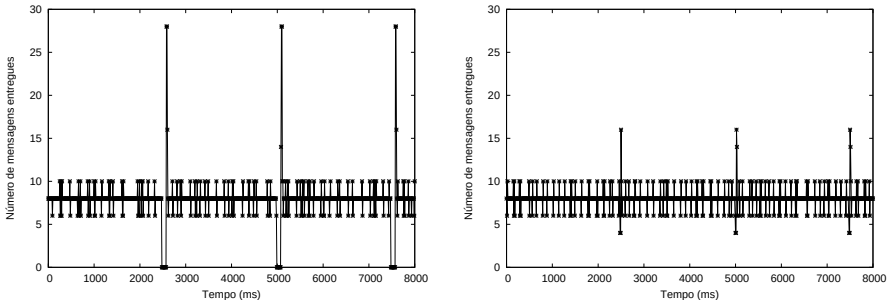
4.6 Comportamento do Sistema Perante um Nó Lento

Até aqui tirámos apenas conclusões usando cenários em que cada nó processa de imediato a ordem de comutação que recebe do coordenador. No entanto, um nó pode sofrer atrasos no processamento das mensagens que recebe. Analisámos a taxa de entrega de mensagens à aplicação, no caso em que um dos nós demora



(a) Genérico (b) Específico

Figura 6. Entrega das mensagens à aplicação - ordem FIFO.



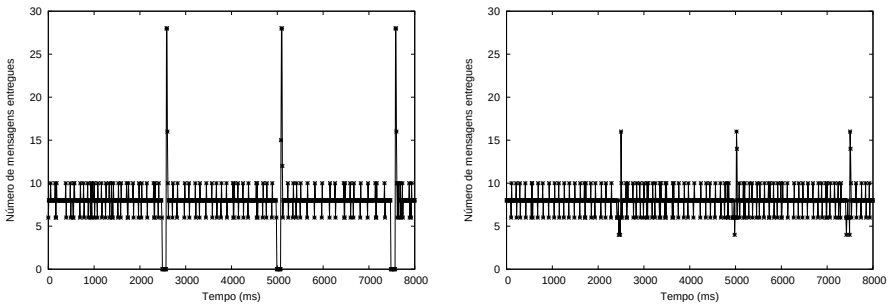
(a) Genérico (b) Específico

Figura 7. Entrega das mensagens à aplicação (nó lento) - ordem total.

1000 ms a processar a ordem para comutar, vinda do *coordenador*. Os resultados são apresentados nas Figuras 7, 8 e 9.

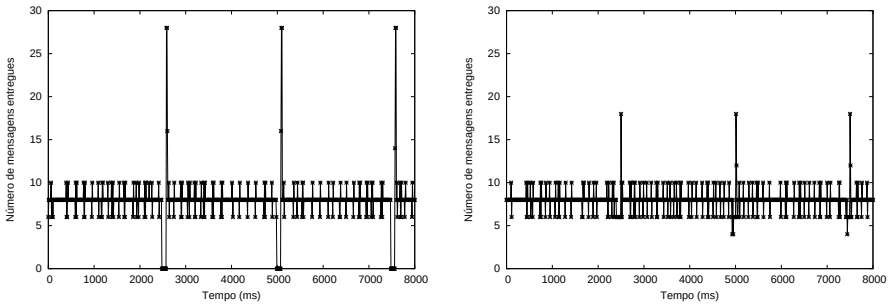
Se compararmos estes resultados com os resultados apresentados anteriormente, observamos que, quando se usa o comutador genérico, a existência de um nó lento faz com que exista um período significativo em que não é entregue nenhuma mensagem à aplicação (enquanto o nó lento não responde). Isto acontece porque o protocolo em causa obriga a que a troca só se efectue quando todos os nós respondem ao grupo dizendo qual o número de mensagens já enviadas utilizando o protocolo A, ou seja, o processo de comutação de um nó está dependente dos outros nós envolvidos na comunicação. Pelo contrário, com os comutadores específicos a interferência para a aplicação é minimizada, uma vez que as únicas mensagens que deixam de ser entregues à aplicação são as mensagens enviadas pelo próprio nó lento, que se acumulam no emissor e depois são enviadas de rajada.

Concluímos então que quando um dos nós é mais lento relativamente aos outros, os protocolos específicos permitem minimizar o impacto que a existência deste nó causa na aplicação distribuída.



(a) Genérico (b) Específico

Figura 8. Entrega das mensagens à aplicação (nó lento) - ordem causal.



(a) Genérico (b) Específico

Figura 9. Entrega das mensagens à aplicação (nó lento) - ordem FIFO.

5 Conclusões

Este artigo abordou o problema de alterar, em tempo de execução, os protocolos de comunicação que suportam a aplicação. Foram propostos dois novos protocolos de comutação específicos, um para suportar a comutação de protocolos de ordem causal e outro para suportar a comutação de ordem FIFO, e foi feita uma análise comparativa do desempenho de protocolos de comutação genéricos e de protocolos de comutação específicos. Os resultados experimentais confirmam e reforçam as expectativas intuitivas: o protocolo genérico interfere significativamente com o fluxo de mensagens. Por outro lado, os protocolos específicos para ordem causal e FIFO, aqui propostos, conseguem suportar a comutação sem afectar o fluxo de mensagens. O protocolo específico para ordem total não evita totalmente a interferência, devido à necessidade de enviar mensagens por dois canais, mas é substancialmente menos intrusivo que o protocolo genérico. As principais razões que suportam este facto são a sincronização entre os nós (que de modo muito particular tem consequência negativas quando um dos nós está atrasado no processamento de mensagens relativamente aos outros), e a necessidade de se armazenarem mensagens mesmo quando as características do

protocolo não obrigavam a tal (como é o caso da ordem FIFO). Há ainda o facto de o protocolo genérico ter de verificar um conjunto de propriedades para poder ser aplicável nos diferentes casos, o que gera processamento e manutenção de estado adicional. Desta forma, pode concluir-se que a utilização de protocolos específicos traz vantagens de desempenho significativas que, do ponto de vista dos autores, justifica o acréscimo de complexidade introduzido pela necessidade de suportar a existência de múltiplos comutadores no sistema.

Agradecimentos Este trabalho foi parcialmente apoiado pela FCT através do projecto "Redico" (PTDC/ EIA/ 71752/ 2006). Os autores agradecem ainda ao José Mocito pelos comentários feitos a versões preliminares deste artigo.

Referências

1. Liu, X., Renesse, R.V., Bickford, M., Kreitz, C., Constable, R.: Protocol switching: Exploiting meta-properties. In: In Proc. 21st IEEE Intl. Conf. on Distributed Computing Systems Workshops (ICDCSW´01), Intl. Workshop on Applied Reliable Group Communication (WARGC). (2001) 37–42
2. Mocito, J., Rodrigues, L.: Run-time switching between total order algorithms. In: In: Proceedings of the Euro-Par 2006. LNCS, Springer-Verlag (2006) 582–591
3. Chen, W.: Constructing adaptive software in distributed systems. In: Proceedings of the 21st International Conference on Distributed Computing Systems, IEEE Computer Society (2001) 635–643
4. Liu, X., van Renesse, R.: Fast protocol transition in a distributed environment (brief announcement). In: PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, New York, NY, USA, ACM (2000) 341
5. van Renesse, R., Birman, K., Hayden, M., Vaysburd, A., Karr, D.: Building adaptive systems using ensemble. Softw. Pract. Exper. **28**(9) (1998) 963–979
6. Rütti, O., Wojciechowski, P.T., Schiper, A.: Structural and algorithmic issues of dynamic protocol update. In: Proceedings of IPDPS '06: the 20th IEEE International Parallel and Distributed Processing Symposium (Rhodes Island, Greece), IEEE Computer Society (2006)
7. Birman, K., van Renesse, R., eds.: Reliable Distributed Computing With the ISIS Toolkit. Number ISBN 0-8186-5342-6. IEEE CS Press (March 1994)
8. Lamport, L.: Time, clocks and the ordering of events in a distributed system. Communications of the ACM **21**(7) (July 1978) 558–565
9. Raynal, M., Schiper, A., Toueg, S.: The causal ordering abstraction and a simple way to implement it. Information processing letters **39**(6) (September 1991) 343–350
10. Goodale, T., Allen, G., Lanfermann, G., Massó, J., Seidel, E., Shalf, J.: The cactus framework and toolkit: Design and applications. In: 5th International Conference In Vector and Parallel Processing (VECPAR), Springer (2003) 182–192
11. Miranda, H., Pinto, A., Rodrigues, L.: Appia: A flexible protocol kernel supporting multiple coordinated channels. In: in Proc. 21st International conference on Distributed Computing Systems (ICDCS-21. (2001) 707–710

A Distributed Bootstrapping Protocol for Overlay Networks*

Miguel Matos and António Sousa and José Pereira and Rui Oliveira
{miguelmatos,als,jop,rco}@di.uminho.pt

Universidade do Minho, Braga, Portugal

Abstract. Peer to peer overlay networks have become popular due to their inherent scalability and resilience properties that come naturally from their decentralized nature. Unfortunately, despite this decentralized approach there is still one important piece that remains centralized: a set of servers to provide identifiers to peers joining the system. This initial step is necessary as new peers need to obtain some contact points in order to establish links to them and join the overlay. This puts out of the protocol model a crucial mechanism as it introduces an external centralized entity to manage the process, leading to problems of scale and fault-tolerance. In scenarios where the churn rate is considerable, the cost of maintaining the list of known peers by the well-know servers may be unbearable. Furthermore, if the peer identifiers provided are not evenly distributed across the universe, this will inevitably cluster the overlay, making it more prone to partitions. In this paper we propose a fully decentralized protocol to obtain those set of initial contact peers using the already deployed infrastructure. By conducting an extensive experimental evaluation we show the effectiveness of the protocol and reason how the ideal number of contact peers may be provided in a fully decentralized fashion.

1 Introduction

Epidemic or gossip-based dissemination protocols present an attractive approach to application level message dissemination in very large scale and dynamic systems. The applicability to these scenarios comes directly from its resilience and scalability, despite its underlying simple principle.

The aforementioned scalability is achieved by spreading the dissemination load among all the participating peers. As all nodes contribute to the dissemination effort, the load imposed on each one only grows logarithmically with the system size, as has been shown in [4]. Another crucial characteristic of gossip-based protocols is the redundancy in the messages received by each node. As peers establish multiple links among them, this leads to a redundant link mesh, the overlay, and consequently to redundancy in the message transmission. This

* This work is supported by HP Labs Innovation Research Award, project DC2MS (IRA/CW118736).

natural redundancy enables the resilience to node and links failures, as multiple embed dissemination trees are always available for each message injected in the system.

As identified in [10] to build an epidemic dissemination algorithm there are several issues that need to be properly addressed: membership, which addresses how peers get to know each other and how many they need to know; network awareness, which deals with reflecting the underlying network topology in the logical connections made among peers; buffer management, which deals with the amount of information each peer needs to keep in order to provide a given reliability level of the dissemination process; and message filtering, whose goal is to take into account the interest of the peers in the dissemination process.

In this paper, we deal with a particular aspect of the membership issue. Although there are multiple protocols that address membership management such as [2,7,11,6,8,3], none of them, to the best of our knowledge, address the problem that arises when a node needs to join the overlay. For this initial joining process to be successful, the joiner needs to know a set of neighbours already on the overlay to which it can connect to. This problem is known as bootstrapping and has been addressed by relying on a set of well-known servers that maintain a list of well-known peers in the overlay. Therefore, upon boot the peer contacts those servers to obtain the set of needed identifiers and then initiates the joining process.

In this paper we propose a fully distributed approach to this problem in the context of the DC2MS [1] project. The goal of the project is to provide a management infrastructure for Cloud environments. The Cloud infrastructure is composed by several data centers spread worldwide and organized in a federation. Therefore, the first goal of the project was to built an adequate membership management and dissemination protocol [9] that is able to provide reliability and scalability guarantees while minimizing the load imposed on the long distance links that connect the several data centers.

As such, we enrich the existing membership management protocol with a distributed bootstrapping mechanism in order to cope with the dynamic and fully distributed scenario as the assumption of a set of well-known poses serious impairments to reliability and scalability and present a single point of failure. Even if the servers are replicated for availability the maintenance of the set of well-known peers is still a problem. In highly dynamic environments such as the ones the DC2MS project targets, peers constantly join and leave the system, and therefore the maintenance of such set of peers may become unbearable, due to the churn factor, or even unattainable in a reliable and scalable fashion. Therefore, we propose a novel fully distributed protocol that removes the necessity of those set of well-known servers as well as the need to known any peer a priori. Our bootstrapping protocol is based on a probabilistic approach and is able to offer an arbitrary set of peers to a joining node. This further increases the usability of the protocol as it allows joining nodes to quickly become indistinguishable of nodes already on the overlay, by establishing multiple links with neighbours since the beginning.

The rest of the paper is organized as follows. In Section 2 we present the membership management protocol previously developed [9] that tackles the network awareness problem. While at first this may seem unrelated to the specific topic addressed in this paper, the goal is to give deeper insight on the way links are established among the peers in the overlay construction protocol and then, from the local knowledge of each peer, proceed to the bootstrapping protocol we propose in Section 3. Furthermore, this unusual approach to the background section is derived from the fact that, to the best of our knowledge, no previous attempt at addressing this problem in a distributed fashion exists. Then, in Section 4, we experimentally evaluate the performance of the protocol and access whether or not it achieves the desired goals. Finally, in Section 5 we conclude the paper.

2 Background

Clon [9] is an overlay management protocol that addresses the network awareness problem when establishing links among the peers participating in the dissemination effort. It builds on the work done in the Scamp protocol [2], due to its natural convergence to the right view size, which makes it adaptable to ever changing system sizes without requiring any global knowledge.

Clon is presented here because the bootstrapping algorithm proposed in this paper and described in Section 3, explicitly takes advantage of the local knownledge available in Clon nodes, as well as the semantics of the overlay management protocol. Nonetheless, the bootstrapping algorithm presented below could be applied to a variety of overlay management protocols such as [2,7,5] as all of them share similar semantics with Clon.

In Scamp joining nodes are integrated into the view of existing peers in a way that the average view size converges to $log(N) + c$, where N is the size of the system and c a protocol parameter related to fault tolerance [4]. The value of this ideal view size has been shown previously in [4]. By integrating nodes with a probability inversely proportional to the view size, and ensuring a given number of subscriptions is sent by the joiner, the protocol is then able to converge to the ideal view size value presented above.

Clon pushes this further by integrating nodes taking into account its locality. By relying on a oracle that should be able to distinguish local nodes from remote ones, the protocol is able to integrate those nodes with different probabilities, effectivelly biasing the overlay to match the underlying network. The mathematic model behind the basic algorithm is well detailed in [2] and therefore we will skip it. The typical applicability scenario of Clon consist of a set of local area networks connected through long-distance wide area networks, where we want to reduce the bandwidth consumption of the dissemination effort.

Listing 1.1 presents the Clon protocol. Upon boot (lines 1 to 4) the new node obtains a contact node from the set of well-known servers, adds the contact to its view and sends an *handleSubscription* request to the contact. Upon receiving

```
1   upon init
2     contact = getContactNode()
3     view.Add(contact)
4     send(contact,handleSubscription(myId))
5
6   proc handleSubscription(nodeId)
7     for n ∈ view
8       send(n,handleJoin(nodeId))
9
10    for i=0; i < c; i++
11      n = randomNode(view)
12      send(n,handleJoin(nodeId))
13
14  proc handleJoin(nodeId)
15    keep = randomFloat(0,1)
16    keep = Math.Floor( localityOracle(viewSize,nodeId) * keep)
17
18    if (keep == 0) and nodeId ∉ view
19     view.Add(nodeId)
20    else
21      n = randomNode(view)
22      send(n,handleJoin(nodeId))
23
24  proc localityOracle(viewSize,nodeId)
25    if isLocal(nodeId)
26      return viewSize * 0.7
27    else
28      return viewSize + viewSize * 0.3
```

Listing 1.1. CLON protocol

the request (lines 6 to 12) the contact node sends an *handleJoin* to all its neighbours and c additional requests to random neighbours.

The nodes that receive the *handleJoin* (lines 14 to 22) then calculate the probability of integration by means of the *localityOracle*, that can be configured in a per-scenario fashion and shall return a positive integer indicating the preference given to that node. Small values will increase the probability of integration while higher values will decrease it. Therefore, by configuring the oracle to return low values to local nodes and high values to remote ones it is possible to effectively bias the overlay to match the underlying topology. Then, in line 16 the value returned by the oracle is weighted against a random value *keep* and, if the outcome is zero the joining node is integrated into the view of the node performing the calculation. If the node is not integrated the *handleJoin* request is forwarded to a random neighbour, in order to preserve the number of links established in the overlay.

2.1 Experimental Evaluation

For the sake of completeness we present an experimental evaluation of the impact of using a network-aware overlay management protocol in the bandwidth consumption on long-distance links such as the ones connection the data centers of a cloud infrastructure.

The simulations presented have been run on a custom simulator in Python. To this end, we used the NETWORKX package which provides graph manipulation facilities to represent the overlay and the gossiping process on top of it. To manipulate the simulation results we recurred to the RPY package which provides a wrapper to the R programming language. The simulation kernel uses discrete time and relies on global state, handling events in a FIFO fashion.

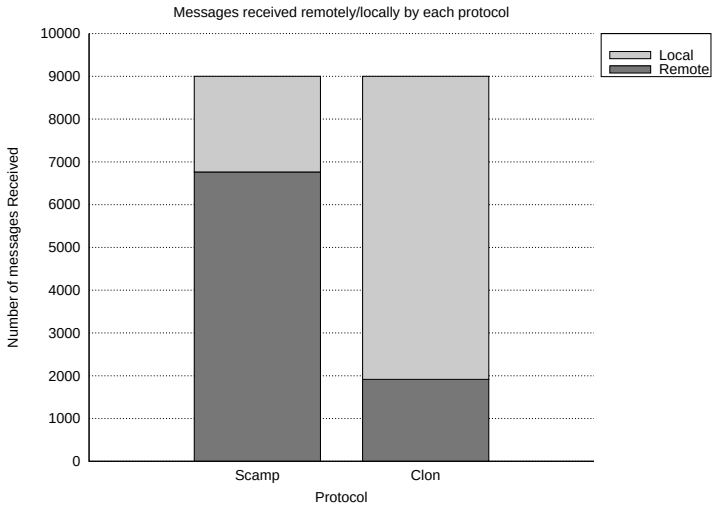


Fig. 1. Messages Received Locally/Remotely by each node.

In the experiment presented here, and in the one presented in the next Section, the scenario consists of 5 local areas with 200 nodes each, accounting for a total of 1000 nodes. Furthermore the c parameter is configured to 6 which yields that the average view size is $log(1000) + 6 = 9$. A detailed analysis of the experimental evaluation of Clon is presented in [9] and is out of the scope of this paper. Here we just present the impact of the number of messages that cross the local and remote links when using the Scamp and Clon protocols. The dissemination protocol used relies on a flooding infect and die dissemination strategy, as it is the most used approach in epidemic dissemination.

The result of this experiment is shown in Figure 1 and consists of having each node inject a new message on the system and then counting the remote and local messages received by each node.

As it is possible to observe, the number of total messages received by each node (local + remote) is 9000 for both protocols. This comes directly from the average node degree, which is 9 and from the number of messages injected on the system 1000, one for each node. If we now look at the number of remote messages it is clear that Clon brings a significant improvement over Scamp, reducing the messages remotely received from around 7000 to 2000, a cut of more that half.

It is important to note that this improvement comes at no penalty in the reliability of the protocol in presence of faults but that analysis is out of the scope of this paper and can be seen in [9]. Furthermore, with a dissemination protocol that also takes into account locality, also presented in [9], it is possible to achieve an improvement of over an order of magnitude in the number of messages that traverse the long-distance links.

For the rest of this paper its is important to keep in mind the way nodes join the overlay in CLON, namely the way the additional c subscriptions are sent as this has a direct impact on the bootstrapping protocol described in the next Section.

3 Bootstrapping Protocol Description

To the best of our knowledge, existing overlay management protocols solve the bootstrapping problem by assuming there is an external entity that provides node identifier(s). This is, in general, not satisfactory as it puts out of the protocol model an important aspect of the overlay building mechanism, and tends to be addressed by relying on static centralized solutions, such as having one or more servers providing the set of initial identifiers. With node churn this set is hard to maintain up to date and provides a brittle solution, even if we made the unrealistic assumption that those servers do not fail. In this proposal we address this problem by fully decentralizing the initial discovery mechanism, making every node in the overlay a potential server in the peer to peer spirit. The only requirement we impose is the availability of a broadcast primitive on the local network where the new node is physically connected. This requirement is virtually guaranteed to be satisfied in modern network architectures, due to the pervasiveness of the TCP/IP communication protocol. As in the CLON protocol described above, the bootstrapping protocol also is network-aware, as it tries to provide the joining node with local and remote peer identifiers. Nonetheless, the solution is flexible enough to be used in other overlay management protocols such as [2,3,5].

Recalling the integration mechanism by which joining nodes get added to the views of other nodes, it is possible to observe that the view size converges, on average, to $log(N)+c$. However, in the the CLON protocol presented in Section 2, a node joining the overlay only establishes one link with the contact node received from the external mechanism, which effectively impairs the connectivity of the joiner. Notwithstanding, if instead of obtaining just one contact node, the joiner obtained 'some' contacts, its connectivity will be improved since the begining. Ideally, this value should be near the average node degree, because in this way the new node will be indistinguishable of nodes that have been on the overlay for more time. Next, we will explain the functioning of the protocol which is depicted in Listing 1.2 and then reason how the ideal result could be achieved.

Upon boot, the node uses the broadcast primitive to request contacts from all the nodes in its local area, as can be seen on lines 3 and 4. If upon reception of the request all nodes replied to the originator, this may led to problems, in a phenomena known as acknowledgement bomb. This phenomena stems from the fact that if every node in a large scale system replies to the originator of a broadcast, the network will become suddenly overloaded, and the requester may be overrun by the amount of replies. To overcome this problem, we rely on an oracle, *sendOracle*, that should instruct whether the node should reply to the request or not, with a given probability. Although the oracle may be configured

```
1   myC = c
2
3   upon init()
4     BCAST(CONTACT, myself)
5
6   proc CONTACT(nodeId)
7     if (sendOracle())
8       if(externalContactOracle(nodeId))
9         contact = randomExternalNode()
10      else
11        contact = myself #or node = randomLocalNode()
12      send(nodeId,(CONTACTREPLY, contact)
13
14  proc CONTACTREPLY(contact)
15    keep = random()
16    keep = Math.Floor((viewSize + 1 *keep)
17
18    if (keep == 0)
19      if view.size() > 0)
20      schedullecCheck()
21      view.Add(contact)
22      send(contact,handleJoin(myId))
23
24      if (myC > 0)
25        send(contact,handleJoin(myId))
26        myC = myC - 1
27
28  #possible send oracle
29  proc sendOracle()
30    totalNodesEstimate = 10^(viewSize-c)
31    localNodesEstimate = totalNodesEstimate / 5 #our scenario has 5 local areas
32
33    if localNodesEstimate < 1
34    localNodesEstimate = 10^viewSize
35
36    reply = viewSize / localNodesEstimate
37    seed = randomFloat(0,1)
38    return seed < reply
39
40  #possible external oracle
41  proc externalContactOracle(nodeId)
42    rand = randomFloat(0,1)
43    return rand < 0.5
44
45  proc cCheck()
46    while(myC >= 0 )
47      contact = randomNode()
48      send(contact,handleJoin(myId))
49      myC = myC -1
```

Listing 1.2. Bootstrapping Protocol

in a naive manner, lets say reply only with a probability of 10%, the amount or replies generated will be effectively reduced, alleviating the problems of the acknowledgement bomb phenomena.

If the oracle instructs the node to reply (line 7), there is another decision that needs to be made: whether to provide a local or remote node as a contact. If this is not taken into account, and joining nodes are always provided with local contacts only, the reliability of the overlay will be compromised. This comes from the fact that if joining nodes only get to know local nodes, over time the number or remote known nodes will decrease considerably and the overlay will partition around the local areas. In the Listing, we show a simplistic oracle on lines 41 to 43 which instructs the protocol to reply half the times with a remote nodes and half of the times with a local one, but naturally this could be configured to the application requirements. After this initial steps where the oracles decide about replying and the kind of node chosen, the contact is sent to the requester (line 12).

Upon reception of a reply (line 14), the joining node should decide whether or not to integrate the contact in its view, based on the same procedure of the original Scamp protocol, where the probability of integration is inversely proportional to the size of the view. The motivation behind this conditional integration is to ensure that even if the oracle of the repliers is not well configured, the view size will still be within the normal bounds of the system. Without this restriction the view size of the joining nodes could become too large and therefore impact the quality of the obtained overlay. If the contact node is to be integrated by the joining node, the latter adds it to its view, i.e. establishes a link with it, and sends it a *handleJoin* as in the Clon protocol presented in the previous Section. With these changes the join mechanism is effectively decentralized and inverted. Instead of being the contact node to send the subscription requests as in Clon, it is now the responsibility of the joining node to send them. This change has an immediate impact on the protocol as the c additional join requests sent to random neighbours must still be transmitted. As such, the joining node becomes the responsible for sending those additional requests. However, instead of sending those additional join request to just one contact, lets say the first received, we decide to distribute them among the several contacts obtained. To this end, the joining node now has an additional variable myC which is initially set to the c protocol parameter, as seen in line 1. Then, for each received contact, the joiner sends the normal *handleJoin* request plus one additional copy until c copies are sent (lines 24 to 26). To prevent the case where less than c contacts are received, and therefore not enough additional copies could be sent, upon the reception of the first contact the protocol schedules the execution of the *cCheck* procedure on a point in the future. This scheduling may be only approximate and should start when it is expected that all the contact replies have been received, which on a local area network shall be close to the first one. This procedure only checks if enough copies have been sent, and if not they are sent to randomly chosen nodes.

After analysing the protocol it is now time to clarify how we could exploit the local knowledge available, in order to obtain optimal results in the bootstrapping mechanism. Optimal in this context means that the joining node establishes as much contacts as the average view size, therefore becoming indistinguishable from other nodes. To achieve this exact behaviour, we will need global knowledge in order to calculate the ideal view size and reply exactly with that amount of contacts to the joiner. Of course this solution is not acceptable, as it will impair all the work done previously on decentralizing the entire protocol. Nonetheless, if we rely only on local knowledge it is still possible to approximate this behaviour, in a probabilistic fashion. In fact, all nodes have a powerful estimation tool of the total amount of nodes, the size of their view. As the view size converges to $log(N)+c$ where a N is the number of nodes in the system it is straightforward to estimate locally the total number of nodes. If we also known the number of local areas available, which probably is well-known (for example the number of data centers of the cloud provider), it is possible to estimate the number of potential repliers to the contact request, i.e. the number of nodes in the local area, and thus reply with the adequate probability. An example of an oracle configured in this way is shown in lines 28 to 38. The oracle estimates the total number of nodes (line 30), calculates the number of local nodes based on this estimation (line 31) and replies with a probability based on this calculations (lines 36 to 38). Although the configuration presented should be well suited to a wide range of scenarios, we still abstract it with an oracle in order to not impair the applicability of the mechanism in other, maybe unpredicted, scenarios. It is important to notice that if the estimation of local nodes is inaccurate, which happens when the view size is inferior to c, the probability of replying adequately will be compromised. This comes from the fact that if the $totalNodesEstimate$ becomes smaller than 1, in the case c is greater than the $viewSize$, then the calculation on line 36 will yield a value greater than 1 and therefore the node will always reply to the contact request. This is easy to observe as $viewSize/localNodesEstimate$ always yields a value greater than 1, when the $localNodesEstimate$ is strictly smaller than 1, which will impair the optimal behaviour we intend to achieve. This abnormality is corrected by ignoring the wrong local nodes estimation and making it $10^{viewSize}$ (lines 33 and 34), which is a rough estimate of the total number of nodes.

4 Experimental Evaluation

In this Section we present the experimental evaluation of the bootstrapping protocol in order to assess if it satisfies the requisite of providing the joining nodes with several contact nodes.

To this end we used different *sendOracle* configurations, starting from a naive one and improving it to obtain the optimal configuration described in the previous Section. The results obtained can be observed in Table 4. The table is organized as follows: the first two columns describe the configuration of the *sendOracle* and *externalContactOracle*, respectively; the third column presents

the number of messages exchanged by the nodes in the runs of the bootstrapping mechanism, without considering the messages sent by the joiners after receiving the contact; in the fourth column it is possible to observe the total number of replies a given joiner obtained; finally the last three columns show the total, local and remote nodes effectively integrated in the view of the joiner.

The scenario is the same as the one described in Section 2 that is we have 5 local areas each one with 200 nodes ammounting for 1000 nodes in the total, the c parameter is set to 6 and as such the ideal view size is 9. For each oracle configuration we run 5000 join operations and extracted the averages of the results obtained. Furthermore, each new run is independent of the previous, i.e. we run the bootstrapping mechanism for a joining node, and after the process ends, we proceed to the next round with a new overlay.

As it is possible to observe for all the configurations the *externalContactOracle* is configured to return true with a probability of 50%, which means that half of the replies will be with local nodes and the other half with remote ones.

The first row of the table shows a naive configuration of the *sendOracle*, where it is configured to always return true. As such, the bootstrap generates 400 messages, 200 for the initial broadcast procedure and 200 replies to the joiner as each node always reply to the requests in this configuration. With this configuration the joiner obtains 200 replies, one for each node on its local area, but only integrates 20 on its view. This is due to the probability of integration being restricted by the actual size of the view, as explained in Section 3. Of this 20 nodes integrated into the view 12 are local and 8 are remote.

On the next configuration, in the second row of the table, the *sendOracle* only replies to the contact requests 5% of the times, which results in sending to the joiner 10 replies ($0.05 * 200 = 10$). Of this 10 replies only 5 are effectively integrated into the view of the joiner, of which 3 are local and 2 remote. The configuration of 5% is just a arbitrary small probability chosen to infer the behaviour of the bootstrapping mechanism.

On the next configuration we start to exploit the local knowledge available in order to approximate the desired optimal behaviour. In this configuration the oracle estimates the total number of nodes in the system, but does not knowns the number of local areas, and as such the probability of replying is only based on the total number of nodes estimated. Nonetheless, the result is interesting as it gets closer to the ideal value of 9 links established by the joining node.

Finally, in the last configuration we exploit the knowledge of the previous configuration but assume that the number of local areas is known beforehand, which may be reasonable for certain scenarios. This configuration corresponds to the example oracle given in Listing 1.2. With this knowledge available, it is possible to achieve the optimal results in the bootstrapping mechanism. In fact, with this configuration the joiner receives 38 contact replies and of those integrates 9 in its view, the ideal value in this scenario. Furthermore, the proportion of local and remote nodes is also closely approximated, as the biasing mechanism of Clon presented previsouly tends to build views with 7 local nodes and 2 remote ones.

To conclude the analysis of the bootstrapping mechanism, the above experiments show that is possible to achieve near optimal configurations with the local knowledge available at each node, as in the third configuration. Furthermore, if the number of local areas of the federation is known beforehand, it is possible to obtain an optimal bootstrapping mechanism that makes joining nodes indistinguishable from the other nodes already present in the overlay.

Oracles Probabilities		Msgs	Contacts	Nodes Integrated		
send	external	Generated	Offered	Total	Local	Remote
1	0.5	400	200	20	12	8
0.05	0.5	210	10	5	3	2
viewSize global	0.5	286	86	13	8	5
viewSize local	0.5	238	38	9	6	3

Table 1. Different bootstrapping configurations.

5 Conclusion

In this paper we proposed a novel fully decentralized bootstrapping mechanism with the goal of removing the requirement of the traditional approach relying on a set of well known servers to provide identifiers to nodes joining the overlay. The advantages of this new bootstrapping mechanism are many fold. First, we eliminate the need to maintain a list of well-known nodes somewhere out of the model, as contact nodes are drawn from all the local nodes on the overlay. As such this also has an impact on the quality of the overlay as contact nodes are chosen more uniformly and therefore the problem of the well-known nodes and its direct neighbours having high degrees is alleviated. This problem draws from the fact that the well-known nodes tend to be frequently contacted by joining nodes and as such they will eventually integrate some of the joiners in their views, hindering the overall node degree distribution. Furthermore, a joining node now knows several initial contact points instead of just one, which effectively improves its connectivity. Finally, the subscriptions along with the c additional copies are sent to different parts of the overlay instead of only the neighbours of the contact node, as in the original protocol, which counters the clustering around those nodes.

As the experimental evaluation attests, the bootstrap protocol is able to provide the joiner with several contact nodes and, taking advantage of the local knowledge available at each node it is possible to achieve optimal configurations that allows joining nodes to become indistinguishable from the rest of the overlay. Nonetheless, if the local knowledge is not available the protocol still achieves satisfactory results, as the experimental evaluation results shown.

Despite the decentralization that could be achieved by the proposed bootstrapping protocol, there is still one important drawback. Although this mechanism works well when the overlay is well established and there are known remote nodes, the initial bootstrap of a whole local area could not be addressed with this mechanism. This is due to the fact that initially no remote nodes are known on the starting local area and as such we still require a set of well-known remote nodes to bootstrap a whole local area, provided by the administrator. Nonetheless, after this initial step the external mechanism could be discarded and, as such our proposal may be used for the rest of the life-cycle with the application with all the advantages we pointed above.

References

1. DC2MS: Dependable Cloud Computing Management Services. http://gsd.di.uminho.pt/projects/projects/DC2MS, 2008.
2. A.-M. K. A. Ganesh and L. Massoulie. SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. In *Networked Group Communication*, pages 44–55, 2001.
3. A.-M. K. A. Ganesh and L. Massoulié. Hiscamp: self-organizing hierarchical membership protocol. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 133–139. ACM, 2002.
4. L. M. A-M. Kermarrec and A. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14:248–258, 2001.
5. T. T. F. Makikawa, T. Matsuo and T. Kikuno. Constructing overlay networks with low link costs and short paths. *Sixth IEEE International Symposium on Network Computing and Applications*, pages 299–304, July 2007.
6. J. P. J. Leitão and L. Rodrigues. HyParView: A membership protocol for reliable gossip-based broadcast. In *IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 419–428. IEEE Computer Society, 2007.
7. A.-M. K. L. Massouli and A. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *In Proceedings of the 22nd Symposium on Reliable Distributed Systems*, pages 47–55, 2003.
8. M. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *In Proceedings of Third European Dependable Computing Conference*, volume 1667 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 1999.
9. M. Matos, A. Sousa, J. Pereira and R. Oliveira. CLON: Overlay Networks and Gossip Protocols for Cloud Environments. Technical Report DI-CCTC-09-13, Centro de Ciências e Tecnologias de Computacão, Universidade do Minho, April 2009.
10. A.-M. K. P. Eugster, R. Guerraoui and L. Massoulie. From epidemics to distributed computing. *IEEE Computer*, 37(5):60–67, May 2004.
11. D. G. S. Voulgaris and M. Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.

Sessão 3C: Computação Móvel e Ubíqua

Algoritmos de Difusão para Protocolos de Encaminhamento em Redes Ad Hoc sem Fios*

João Matos e Hugo Miranda

LaSIGE/Universidade de Lisboa

Resumo A entrega de uma mensagem a todos os participantes (difusão) é uma peça fundamental em muitos dos protocolos de encaminhamento reactivos para redes ad hoc. O algoritmo de difusão mais utilizado é o de "inundação", que apresenta um consumo de recursos excessivo. Este artigo compara o desempenho do protocolo de encaminhamento AODV utilizando três algoritmos de difusão distintos. É avaliado o seu impacto na redução do tráfego na rede e na qualidade das rotas utilizadas.

1 Introdução

A comunicação entre os participantes numa rede sem fios é normalmente assegurada por uma infra-estrutura de suporte. A infra-estrutura é responsável pelas funções de gestão, como a atribuição de endereços, e pela prestação dos serviços básicos de comunicação, como o escalonamento das transmissões dos dispositivos e o encaminhamento das mensagens. A instalação destas redes não é instantânea, pois é necessária a colocação e configuração das estações base.

No entanto, existem situações em que a instalação antecipada da infra-estrutura não é possível, por exemplo porque a sua relevância não pode ser antecipada. Exemplos são as operações de busca e salvamento e operações militares. Neste tipo de cenários, e devido à impossibilidade de instalação da infra-estrutura, os serviços que esta disponibiliza terão que passar a ser assegurados exclusivamente pelos dispositivos participantes. As redes sem suporte da infra-estrutura são usualmente denominadas redes ad hoc móveis ou *MANETs*.

Neste trabalho, assumimos que as redes ad hoc são compostas por utilizadores cooperantes e que por isso disponibilizam os seus dispositivos para que participem na gestão da rede. Tal não implica contudo que os dispositivos permaneçam imóveis. Os interfaces rádio dos dispositivos móveis têm um raio de transmissão limitado, que pode ser insuficiente para alcançar todos os participantes. Nas redes ad hoc, cabe aos dispositivos geograficamente localizados entre o emissor e o destinatário facilitar a entrega, retransmitindo a mensagem. Para além da retransmissão, inclui-se nas funções de gestão a realizar pelos participantes a descoberta de uma sequência de nós, vulgarmente designada de rota, que assegure a comunicação. Este problema tem alguma complexidade uma vez que

* Este trabalho foi parcialmente suportado pelo projecto PTDC/EIA/71752/2006, REDICO: Dynamic Reconfiguration of Communication Protocols e pelo programa de financiamento plurianual.

é assumido que os participantes podem não ter conhecimento da sua localização geográfica ou da dos destinatários. Adicionalmente, pretende-se descobrir uma rota com o menor custo, usualmente avaliado pelo número de intermediários.

Alguns dos protocolos de encaminhamento para redes ad hoc mais populares (por exemplo [1,2]) utilizam um algoritmo de "inundação" (do inglês *flooding*) durante o processo de descoberta da rota. Neste caso, o flooding consiste na difusão de uma mensagem pelo emissor indicando o destinatário pretendido. Esta mensagem é propagada por todos os participantes. Cada dispositivo irá por isso em princípio receber uma cópia da mensagem de cada vizinho. Ao receber a primeira cópia, o participante responde ao emissor se for o destinatário ou se conhecer uma rota para o destinatário. Caso contrário, retransmite a mensagem para que ela seja recebida por todos os seus vizinhos.

Intuitivamente, e a menos de problemas ao nível físico, como interferências, o flooding assegura a entrega da mensagem a todos os participantes que se encontrem na partição de rede do emissor.[1] Contudo, na maior parte das situações, a retransmissão por todos os participantes representa um consumo excessivo de recursos uma vez que o mesmo objectivo pode ser assegurado com a retransmissão de apenas alguns participantes, localizados em pontos chave da partição. Adicionalmente, o elevado número de retransmissões gera colisões que contribuem para a perda de largura de banda útil e para o aumento do consumo de energia, num processo conhecido como *broadcast storm* [3].

A gestão eficiente dos recursos é uma temática central da investigação em redes ad hoc, sabendo-se o forte peso das interfaces de rede no consumo energético dos dispositivos [4]. Contudo, o flooding é utilizado com frequência em protocolos para MANETs. Recentemente, têm vindo a ser estudados diferentes algoritmos de difusão. O objectivo é diminuir o número de dispositivos que retransmitem cada mensagem, reduzindo desta forma o consumo energético das operações de difusão. No entanto, estes algoritmos podem apresentar resultados sub-óptimos quando aplicados em substituição do flooding. Neste artigo analisamos o caso particular da aplicação de um algoritmo de difusão ao AODV [2], um dos protocolos de encaminhamento mais populares para redes ad hoc.

O artigo está estruturado da forma descrita em seguida. Na Sec. 2 apresentam-se as alternativas ao flooding em estudo e uma panorâmica sobre o funcionamento do protocolo de encaminhamento utilizado. A Sec. 3 aborda o problema da integração dos algoritmos de difusão com o protocolo de encaminhamento. A comparação do desempenho das diferentes combinações é apresentada na Sec. 4. Finalmente a Sec. 5 conclui o artigo.

2 Trabalho Relacionado

2.1 Algoritmos de Difusão

O objectivo dos algoritmos de difusão é a entrega de mensagens ao maior número de participantes possível, idealmente a todos os dispositivos na partição onde a

[1] Uma partição é definida pelo conjunto de dispositivos que se encontrem no raio de transmissão de pelo menos um outro membro dessa partição.

disseminação da mensagem foi iniciada. Espera-se que o consumo de recursos, ditado sobretudo pelo número de retransmissões da mensagem, bem como a latência entre a produção da mensagem e a sua entrega ao último dispositivo sejam tão reduzidos quanto possível.

Uma das aproximações mais populares para reduzir o número de transmissões consiste em aplicar funções probabilistas localmente a cada dispositivo. Estas funções permitem que apenas alguns dos dispositivos retransmitam sem que para tal seja necessária a coordenação dos dispositivos.

Algoritmos Probabilistas Os algoritmos de difusão probabilistas são regulados por um parâmetro de configuração p que representa a probabilidade de um participante retransmitir uma mensagem quando a recebe pela primeira vez.[2]

Os algoritmos baseados exclusivamente numa probabilidade de retransmissão não são facilmente adaptáveis a diferentes condições da rede. Embora a probabilidade p permita estimar a proporção de dispositivos que retransmitirá numa dada região, não permite estimar o número absoluto de retransmissões. Este dependerá do número absoluto de vizinhos. Pode por isso acontecer que todos os participantes decidam não retransmitir. Embora este evento seja bastante mais provável quando se utiliza uma probabilidade de retransmissão moderada em regiões onde o número de vizinhos é reduzido, pode resultar numa terminação precoce da difusão, ou seja, sem que a mensagem tenha sido entregue a todos os participantes. De notar que a frequência da ocorrência destes eventos pode ser diminuída aumentando a probabilidade de retransmissão. No entanto, uma probabilidade de retransmissão mais elevada resulta num número excessivo de retransmissões em zonas onde a concentração de dispositivos é elevada,

O algoritmo GOSSIP3(p, k, m) [5] propõe atenuar este problema mantendo uma probabilidade de retransmissão moderada mas enriquecendo o algoritmo probabilista com mecanismos que permitem detectar e reagir a algumas condições de excepção. A probabilidade de retransmissão p é complementada com critérios adicionais de retransmissão, regulados pelos parâmetros k e m. O parâmetro k visa favorecer a propagação da mensagem na fase inicial, momento de particular vulnerabilidade dado o número reduzido de dispositivos que recebeu a mensagem. Para tal, a mensagem inclui um contador inicializado a zero e incrementado por cada dispositivo que efectua a retransmissão. Os dispositivos que recebam a mensagem com o contador com valor inferior a k retransmitem obrigatoriamente a mensagem.

O parâmetro m assegura um número mínimo de retransmissões na vizinhança de qualquer dispositivo. Os dispositivos que decidem não retransmitir em resultado da função probabilista retêm a mensagem numa fila. A mensagem será retransmitida pelo dispositivo se após um período de tempo predefinido o número de cópias recebidas for inferior a m.

Apesar de assegurar um número mínimo de retransmissões, a selecção dos dispositivos que retransmitem no algoritmo GOSSIP3(p, k, m) é aleatória. Para

[2] O algoritmo de flooding pode ser visto como um caso particular de algoritmo probabilista onde $p = 1$.

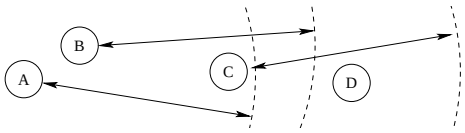


Figura 1. Raio de transmissão de 3 dispositivos

demonstrar o impacto da selecção dos dispositivos na eficiência dos algoritmos de difusão, considere-se o exemplo representado na Fig. 1, onde um dispositivo A retransmite uma mensagem, entregando-a aos dispositivos B e C. Apesar de ambos os dispositivos apresentarem uma probabilidade idêntica de retransmissão, o impacto da retransmissão de cada um é distinto. A retransmissão por B é sempre desnecessária por não contribuir com a entrega da mensagem a qualquer outro dispositivo. Este efeito será tanto mais observável quanto a proximidade de A e B for maior. Adicionalmente, importa notar que se C não retransmitir, a mensagem poderá não ser entregue ao dispositivo D.

Os mecanismos introduzidos no algoritmo GOSSIP3(p, k, m) não garantem a propagação da mensagem para D se o algoritmo for configurado com $k = 1$. O mesmo efeito poderá ser observado para outros valores do parâmetro k se assumirmos a existência de outros dispositivos no raio de transmissão de A. Foi já demonstrado que problemas similares são observáveis em outros algoritmos que realizam a selecção probabilista dos dispositivos [6].

Algoritmos Baseados em Distância O exemplo da Fig. 1 sugere que os melhores candidatos à retransmissão são os dispositivos que se encontram mais afastados daqueles que realizaram as retransmissões anteriores. O PAMPA [7] é um algoritmo de difusão que substitui a aleatoriedade na selecção pela distância entre dispositivos. Para evitar operações de coordenação, cada dispositivo atrasa a sua retransmissão por um tempo dado por uma função *delay* que recebe como argumento a força de sinal (RSSI) com que a mensagem foi entregue ao dispositivo. Durante o tempo de espera correspondente ao atraso determinado, o dispositivo contabiliza o número de cópias da mensagem recebidas e a retransmissão não terá lugar se este número atingir um valor pré-definido. Caso contrário ocorre quando o tempo de espera imposto expirar.

A função *delay* típica multiplica o valor da intensidade da força de sinal por uma constante pré-definida, convertendo-a num intervalo de tempo. Uma vez que a força do sinal decai significativamente com o aumento da distância entre dois participantes, os dispositivos que se encontram mais distantes aplicarão um atraso menor. Atendendo à relação entre o atraso e a distância, descartar uma retransmissão significa que a difusão já foi assegurada por outros participantes, melhor posicionados.

Dentro do raio de alcance de cada participante emissor, o PAMPA vai escolher os participantes que se encontram mais afastados. Portanto vai seleccionar, para cada mensagem difundida, o grupo de participantes cuja localização faz uma maior cobertura, prevendo-se uma utilização de retransmissões bas-

tante menor, relativamente a outros algoritmos de difusão, como o flooding e o GOSSIP3(p, k, m).

2.2 AODV

O Ad hoc On-demand Distance Vector (AODV) [2] é um protocolo de encaminhamento para redes ad hoc móveis. No AODV todos os participantes podem assumir o papel de encaminhador e mantêm uma tabela de encaminhamento. Para cada destinatário conhecido, a tabela de encaminhamento armazena o dispositivo seguinte na rota.

O AODV é um protocolo reactivo. Ou seja, as tabelas de encaminhamento são povoadas durante as operações de descoberta de rota, iniciadas pelos dispositivos sempre que necessitam de uma rota para um dispositivo que não consta da sua tabela. Se o participante emissor não possuir nenhuma informação sobre o destinatário na sua tabela de encaminhamento, difunde uma mensagem de pedido de rota (*route request*). Na versão original do AODV, o *route request* é difundido utilizando o algoritmo de flooding.

A mensagem de *route request* inclui, entre outros, campos para os endereços do emissor e do destinatário e um contador para o número de saltos (incrementado a cada retransmissão). Dois outros campos (número de sequência do emissor e número de sequência da última rota conhecida para o destinatário) permitem evitar a utilização de rotas desactualizadas. Os dispositivos podem identificar duplicados de pedidos de rotas pelo par <endereço do emissor,identificador de broadcast> de cada mensagem.

Quando um participante recebe uma mensagem de *route request* pela primeira vez, começa por verificar se possui na sua tabela de encaminhamento uma rota para o destinatário, caso em que vai responder ao emissor com uma mensagem de resposta (*route reply*). A retransmissão da mensagem ocorre apenas no caso contrário. As mensagens de resposta são também produzidas pelo destinatário quando este recebe um *route request*. Quando o canal de comunicação é bidireccional, os *route reply* são enviados ponto-a-ponto percorrendo a sequência inversa dos dispositivos que retransmitiram o *route request*.

De notar que as rotas são criadas no sentido inverso ao da propagação das mensagens de *route request* e *route reply*. Ou seja, ao receber de um dispositivo A uma mensagem de pedido de rota produzida por um dispositivo B, o dispositivo regista na sua tabela de encaminhamento que as mensagens com destino a B devem ser entregues ao dispositivo A. Analogamente, o dispositivo que recebe de A uma resposta de rota para o dispositivo B regista na sua tabela de encaminhamento que A é o próximo salto das mensagens destinadas a B.

A mobilidade dos participantes pode tornar inválidas rotas adquiridas no passado. As mensagens de erro na rota (*route error*) são geradas por dispositivos intermédios incapazes de entregar uma mensagem de dados ao dispositivo que consta da sua tabela de encaminhamento. Na maior parte dos casos, a reacção à recepção de uma destas mensagens pelo emissor da mensagem de dados resultará na difusão de um novo pedido de rota.

É previsível que uma mensagem de pedido de rota continue a ser difundida pela rede após o participante emissor ter obtido resposta. Este problema é realçado quando o participante emissor e o participante destinatário (ou um participante que possua o próximo salto da rota em causa) se encontrem próximos numa rede geograficamente dispersa. Para atenuar este problema, o AODV difunde a mensagem de pedido de rota através de um *expanding-ring search*. Esta optimização consiste em limitar inicialmente o número de saltos do *route request* de forma a que ela seja retransmitida apenas pelos participantes que se localizam na proximidade do emissor. Se não obtiver resposta após algum tempo, o processo é repetido com um número de saltos limite maior, tipicamente suficiente para que seja recebida por todos os dispositivos na rede.

3 Alternativas à Inundação no AODV

A difusão das mensagens de descoberta de rota é uma componente fundamental do AODV sem a qual não seria possível assegurar a comunicação entre quaisquer dois dispositivos. A capacidade dos dispositivos se deslocarem livremente, aliada ao limitado raio de transmissão tornam as operações de descoberta de rota frequentes mesmo em cenários onde o número de canais de comunicação estabelecidos entre os participantes é limitado. Reduzir o custo das operações de difusão contribui por isso para aumentar a longevidade das baterias dos dispositivos e a largura de banda útil.

Contudo, a utilização do flooding apresenta um conjunto de propriedades importantes para a definição das rotas. Uma operação de flooding permite descobrir a totalidade das rotas entre quaisquer dois dispositivos, uma vez que todos os participantes retransmitem a mensagem. Tão grande número de combinações é claramente desnecessário uma vez que muitas apresentarão custos claramente excessivos. O AODV evita a descoberta dessas rotas impondo a cada dispositivo uma única retransmissão da mensagem. Intuitivamente, a primeira cópia do *route request* recebida por cada dispositivo será aquela que viaja por uma rota mais favorável, com menor congestionamento e menor número de saltos. Com o modelo de propagação base do AODV, cada pedido de rota define um conjunto de rotas, todas elas disjuntas nos dispositivos intermédios e que incluirá sempre a rota mais favorável.

Na prática, esta propriedade poderá não se verificar devido às perturbações introduzidas ao nível de ligação de dados pelo número de transmissões concorrentes impostas pelo flooding. Dependendo da política de controlo de acesso ao meio utilizada, transmissões concorrentes poderão gerar colisões fazendo com que algumas das melhores rotas não sejam descobertas. Para evitar este problema, aplica-se tipicamente um atraso aleatório à retransmissão das mensagens e que pode igualmente prejudicar (artificialmente) as melhores rotas.

Os algoritmos de difusão alternativos contrastam com o flooding. Por um lado, a selecção aleatória dos dispositivos que retransmitem não assegura que seja encontrada a melhor rota, uma vez que os dispositivos que a compõem podem não ser seleccionados. Por outro, não deverão apresentar problemas de congestão

tão severos, beneficiando os tempos de descoberta de rotas e aumentando a largura de banda útil. Importa por isso comparar o desempenho dos protocolos de encaminhamento utilizando alguns dos algoritmos de difusão da literatura.

Uma primeira aproximação foi apresentada em [5], que compara o desempenho do AODV utilizando o flooding e o GOSSIP3(p, k, m). O estudo dos autores concluiu que a combinação de parâmetros GOSSIP3$(0.65, 1, 1)$ seria a que apresentava um desempenho mais vantajoso para um conjunto de topologias genéricas. De notar que na substituição do flooding pelo GOSSIP3(p, k, m) não são introduzidos atrasos suplementares nem é aplicada qualquer política para a selecção dos dispositivos que irão retransmitir.

Os resultados da comparação sugerem que a substituição é benéfica [5]. Os autores observaram que substituindo o flooding pelo GOSSIP3$(0.65, 1, 1)$ no AODV, o número de retransmissões de pedidos de rota é reduzido até 35%. Esta redução do número de retransmissões tem impacto directo no desempenho do AODV, ao libertar largura de banda e reduzir o número de colisões, aliviando o problema de *broadcast storm*. Em resultado, verificou-se uma melhoria da taxa de entrega e da latência.

Foi também observado que as rotas obtidas são entre 10% e 15% mais longas do que as rotas obtidas pelo flooding. O comprimento das rotas obtidas pelo GOSSIP3$(0.65, 1, 1)$ tende a aproximar-se dos resultados do flooding à medida que se aumenta a probabilidade de retransmissão [5]. Este resultado pode ser explicado por um aumento da proporção de dispositivos que retransmitem a mensagem (representada por um aumento na probabilidade de retransmissão) implicar um acréscimo na probabilidade de que uma das retransmissões seja realizada pelo dispositivo que seria seleccionado pelo flooding.

3.1 AODV+PAMPA

Ao contrário do GOSSIP3(p, k, m), o PAMPA impõe uma política de selecção dos dispositivos que retransmitem cada mensagem bem como um atraso adicional à propagação. Na versão do AODV que utiliza o PAMPA como mecanismo de difusão, quando um participante recebe um pedido de rota, verifica se dispõe de uma rota para o destinatário na sua tabela de encaminhamento ou se a mensagem já foi recebida. Em ambos os casos respeita-se o protocolo AODV original, respectivamente respondendo à mensagem com um *route reply* ou ignorando-a. À primeira cópia de um *route request* para o qual não se dispõe de rota na tabela de encaminhamento é aplicada a política de retransmissão do PAMPA. Ou seja, a mensagem é colocada em espera numa fila durante o intervalo de tempo calculado pela função *delay*. Se durante esse tempo o participante receber uma cópia da mensagem em causa, vai removê-la da fila, bem como descartar a cópia que recebeu. Caso contrário, a mensagem é retransmitida quando o temporizador expirar.

Dentro do alcance de cada emissor, o PAMPA escolhe os participantes que se encontram mais afastados. Prevê-se portanto, que utilizando o PAMPA, o AODV apresente um menor número de retransmissões de *route requests*. Adicionalmente, o AODV apresentará na maior parte dos casos a rota mais curta.

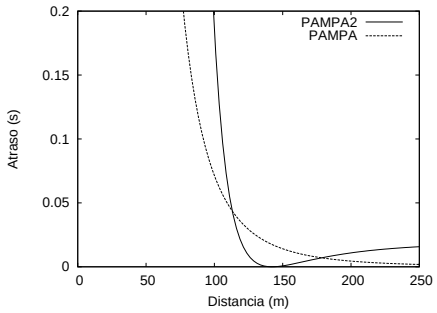


Figura 2. Comparação do atraso aplicado pelas funções *delay* e $delay_2$

Um cenário partilhado com o flooding mas não com o GOSSIP3(p, k, m) uma vez que neste, a selecção aleatória possibilitará a definição de rotas com maior proximidade entre os dispositivos.

Um número menor de saltos contribui para uma menor ocupação do meio por implicar um número menor de retransmissões de mensagens. No entanto, a utilização do PAMPA favorece um aumento da frequência da quebra de rotas. Uma vez que os dispositivos que integram uma rota estão o mais distantes possível (limitado pelo raio de transmissão), os seus movimentos irão mais facilmente colocá-los fora do alcance dos restantes. Nestes casos, a rota deixa de ser válida e consequentemente tem de ser repetido o processo de descoberta de rota o que implica um aumento do número de retransmissões.

Para atenuar este problema, foi avaliada uma segunda função *delay*, designada $delay_2$ que privilegia dispositivos ligeiramente mais próximos da origem. Ou seja, em vez de seleccionar o grupo de participantes que faz a maior cobertura, a função $delay_2$ selecciona preferencialmente um grupo menos sujeito a quebra de rotas e que ainda proporciona uma cobertura bastante aceitável.

Ao invés da função $delay(x)$ que é da forma $delay(x) = kx$ e portanto linear, a função $delay_2$ é quadrática, colocando o seu vértice no interior do raio de transmissão dos dispositivos. Uma desvantagem da função $delay_2$ é a necessidade de se saber antecipadamente qual o raio de transmissão dos dispositivos por forma a determinar os diferentes coeficientes. Uma função adequada foi determinada experimentalmente para a placa de rede IEEE 802.11 utilizada no simulador de redes ns–2 e que define um raio de transmissão de 250m. Na avaliação foi utilizada a função $delay_2(x) = \frac{4}{25}(x + 1.5)^2 - \frac{8}{5}(x+1.5)+4$. A Fig. 2 mostra o atraso aplicado pelas funções *delay* e $delay_2$ em função da distância entre os dispositivos. De notar que por clareza, se utiliza PAMPA e PAMPA2 para distinguir as duas versões da função.

4 Avaliação

O desempenho do AODV utilizando os algoritmos de flooding, GOSSIP3$(0.65, 1, 1)$, PAMPA e PAMPA2 foi comparado através de simulações preparadas utilizando

o simulador de rede ns–2. O objectivo é perceber qual o impacto do algoritmo de difusão no tráfego gerado e na qualidade das rotas seleccionadas. Os cenários avaliados, descritos em seguida, foram baseados nos utilizados em [5] para comparar o desempenho do AODV utilizando GOSSIP3(p, k, m) e flooding.

Em todas as simulações 150 dispositivos utilizando uma interface de rede IEEE802.11 com um raio de transmissão de 250m são inicialmente dispostos de forma aleatória numa área de $3300m \times 600m$. Em cada simulação são estabelecidas 30 ligações entre dispositivos seleccionados aleatoriamente. Cada uma gera 2 pacotes de 512 bytes por segundo. As simulações têm uma duração de 525s.

Os dispositivos deslocam-se de acordo com o modelo de movimento *random waypoint* [8], a uma velocidade uniforme compreendida entre $0ms^{-1}$ e $20ms^{-1}$. Foram experimentados tempos de pausa de 0s, 100s, 300s e 500s, definindo assim 4 cenários de mobilidade.

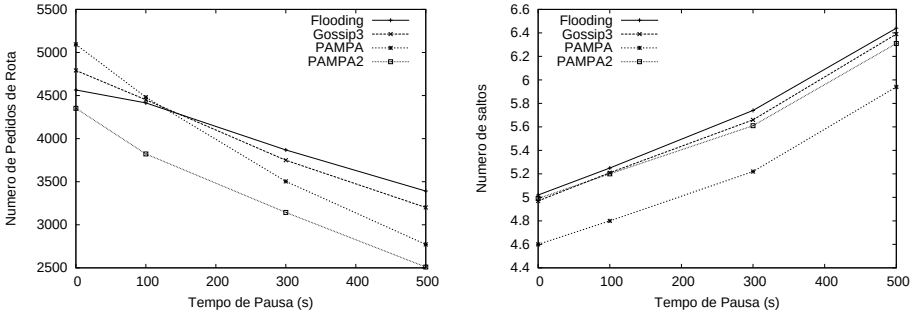
A concretização base do AODV no simulador ns–2 foi adaptada por forma a utilizar cada um dos algoritmos de difusão estudados, nomeadamente flooding, GOSSIP3$(0.65, 1, 1)$, PAMPA e PAMPA2. Estes últimos configurados para descartar as mensagens após receberem uma retransmissão. A concretização do AODV define 2 passos preliminares para o *expanding-ring search*, respectivamente com 5 e 7 saltos. A difusão das mensagens durante o *expanding-ring search* é também assegurada pelo algoritmo de difusão em avaliação.

Para cada cenário de mobilidade foram geradas aleatoriamente 40 simulações com movimentos e tráfego distintos. São aplicadas as mesmas simulações a todos os algoritmos. Os valores apresentados são a média aritmética dos resultados das 40 simulações de cada algoritmo em cada cenário.

Estabilidade das Rotas O número de pedidos de rotas solicitados pelos dispositivos em cada um dos algoritmos é apresentado na Fig. 3(a). Tal como esperado, o aumento do tempo de pausa resulta numa maior durabilidade das rotas. Uma vez que uma das razões que origina um pedido de rota é a quebra daquela que se encontrava em utilização, todos os algoritmos apresentam um decaimento do número de pedidos de rota, em linha com o aumento do tempo de pausa. A figura confirma também as suspeitas levantadas anteriormente sobre a qualidade das rotas seleccionadas por cada algoritmo, em particular, a baixa tolerância do PAMPA ao movimento dos dispositivos. Para os tempos de pausa mais baixos, o PAMPA apresenta a maior taxa de quebra de rotas e consequentemente de renovação de pedidos. Esta vulnerabilidade do PAMPA é confirmada na Fig. 3(b), que apresenta a média de saltos de todas as mensagens de resposta aos pedidos de rota. O PAMPA apresenta aqui as rotas mais curtas, o que confirma que são os participantes mais afastados do emissor que retransmitem.

Os bons resultados da função $delay_2$ (PAMPA2) ao nível do número de pedidos de rotas sugerem que a função é adequada para a selecção das rotas. No entanto, a Fig. 3(b) confirma que a estabilidade das rotas é conseguida com um número de saltos mais elevado, próximo dos restantes algoritmos.

A tendência do modelo de movimento *random waypoint* em concentrar os dispositivos no centro da área simulada [9] é, nestas simulações, claramente observável na Fig. 3(b) onde, para uma região com a mesma dimensão e para



(a) Operações de descoberta de rota (b) Comprimento médio das rotas

Figura 3. Qualidade das rotas

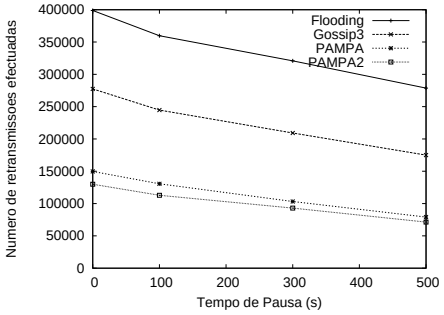


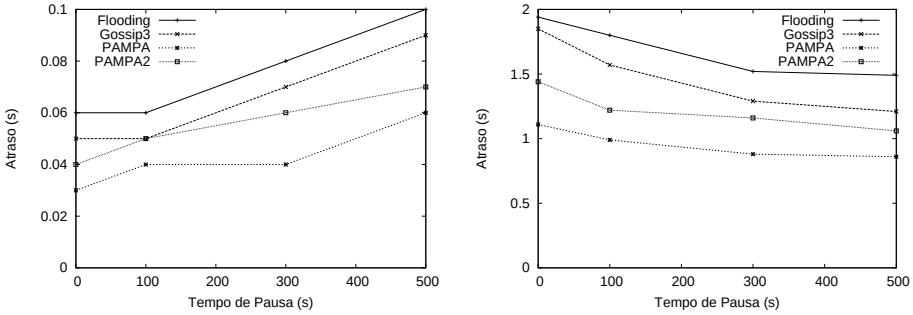
Figura 4. Retransmissões de pedidos de rota

dispositivos com o mesmo alcance de transmissão, os tempos de pausa superiores tendem a apresentar rotas com maior número de saltos.

Tráfego A Fig. 4, que apresenta o acumulado de retransmissões de pedidos de rota realizadas por todos os dispositivos mostra uma distinção clara entre os algoritmos em estudo. O PAMPA concretiza reduções de tráfego até 72% e o PAMPA2 até 74%, em relação ao flooding. Em comparação com o GOSSIP3$(0.65, 1, 1)$, os ganhos atingem 55% para o PAMPA e 59% para o PAMPA2.

O gráfico confirma também a relevância dos algoritmos de difusão no desempenho das MANETs uma vez que o número total de retransmissões de pedidos de rota diminui nos cenários onde (segundo a Fig. 3(b)) as rotas são mais longas. Ou seja, nos cenários apresentados, a parte mais significativa do tráfego total pertence às operações de descoberta de rotas e não à transmissão dos dados.

Latência A redução do tráfego gerado pelos algoritmos PAMPA e PAMPA2 nas operações de descoberta de rota é utilizada para justificar a latência mais reduzida apresentada por estes algoritmos na Fig. 5. Ao reduzir o número de



(a) Operações de descoberta de rota (b) Entrega de dados

Figura 5. Latência

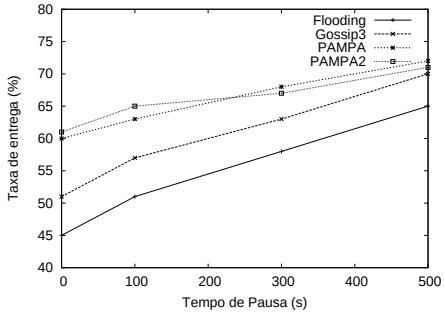


Figura 6. Taxa de entrega

dispositivos que retransmitem, estes algoritmos contribuem para diminuir os tempos de contenção, activados pelo nível de ligação de dados para atenuar a congestão. Ou seja, as figuras comprovam que os atrasos explícitos impostos pelos algoritmos PAMPA são atenuados pelo alívio de tráfego resultante.

Taxa de entrega A Fig. 6 termina a análise aos resultados das simulações apresentando a taxa de entrega das mensagens de dados. A clara vantagem do PAMPA e do PAMPA2 nos tempos de pausa mais reduzidos é uma vez mais atribuída ao reduzido tráfego produzido pelas operações de descoberta de rotas em comparação com os restantes algoritmos.

Esta perspectiva é confirmada pela convergência evidenciada para tempos de pausa mais elevados, situação em que os restantes algoritmos apresentam diminuições de tráfego significativas com a consequente diminuição do número de colisões a ocorrer na rede.

O PAMPA e o PAMPA2 apresentam taxas de entrega muito semelhantes. Nos cenários com maior movimento, o PAMPA tem uma taxa de entrega ligeiramente mais reduzida do que o PAMPA2 por ser mais sujeito a quebra de rotas.

5 Conclusões

Os protocolos de encaminhamento que utilizam o flooding como algoritmo de difusão, como o AODV, sobrecarregam a rede com retransmissões excessivas, consumindo desnecessariamente recursos dos dispositivos e largura de banda. Algoritmos probabilistas, como o GOSSIP3(p, k, m), contribuem para uma atenuação deste problema mas apresentam limitações na adaptação dinâmica a diferentes condições de rede. Este artigo analisou as vantagens da aplicação do algoritmo de difusão PAMPA a protocolos de encaminhamento. As simulações realizadas demonstram que as características únicas do PAMPA contribuem para uma redução ainda mais significativa do tráfego. Contudo, estas mesmas características tornam o AODV menos resiliente ao movimento dos dispositivos. O artigo propõe o algoritmo de difusão PAMPA2 que atenua esta limitação do PAMPA, apresentando por isso um melhor desempenho em cenários com elevado movimento.

Referências

1. 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
2. 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
3. Tseng, Y.C., Ni, S.Y., Chen, Y.S., Sheu, J.P.: The broadcast storm problem in a mobile ad hoc network. Wireless Networks **8**(2/3) (2002) 153–167
4. Feeney, L.M., Nilsson, M.: Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In: Procs. of the 20th Conf. of the IEEE Comp. and Comm. Soc. (INFOCOM 2001). Volume 3. (2001) 1548–1557
5. Haas, Z.J., Halpern, J.Y., Li, L.: Gossip-based ad hoc routing. In: Procs. of the 21st Joint Conf. of the IEEE Comp. and Comm. Societies (INFOCOM 2002). Volume 3. (2002) 1707–1716
6. Miranda, H., Leggio, S., Rodrigues, L., Raatikainen, K.: Removing probabilities to improve efficiency in broadcast algorithms. In: 5th MiNEMA Workshop. (2007)
7. Miranda, H., Leggio, S., Rodrigues, L., Raatikainen, K.: A power-aware broadcasting algorithm. In: Procs. of The 17th IEEE Int'l Symposium on Personal, Indoor and Mobile Radio Comm. (PIMRC'06). (2006)
8. Johnson, D.B., Maltz, D.A.: Dynamic Source Routing in Ad Hoc Wireless Networks. In: Mobile Computing. Kluwer Academic Publishers (1996) 153–181
9. Bettstetter, C., Resta, G., Santi, P.: The node distribution of the random waypoint mobility model for wireless ad hoc networks. IEEE Trans. on Mobile Computing **2**(3) (2003) 257–269

FEW Phone File System

João Soares and Nuno Preguiça

CITI/Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

Abstract. Data availability is an important issue in mobile computing environments. Both distributed data management systems and mobile storage devices, such as USB-flash drives, are used to allow users ubiquitous access to their data, but both solutions have their shortcomings.
In this paper, we present the FEW Phone File System, a system designed to allow users to access their personal data independently of user's location or machine's connectivity. By combining the advantages of both mobile storage devices and distributed data management systems, the FEW Phone File System allows users to carry replicas of their data on their mobile phones, and access them as regular files in a file system, by relying on the wireless communications capabilities of these devices.
To deal with the limitations of these devices, the FEW Phone File System integrates mechanisms that allows multimedia contents to be adapted to the mobile phone's specifications, thus using much less storage. Additionally, the system includes a data source verification mechanism that keeps track of additional sources for each file (e.g. web, CVS/SVN repositories), thus allowing for these remote sources to be used as source alternatives for the user's data.

1 Introduction

Today's users are no longer confined to a single personal computer, instead they tend to have multiple personal computers. To be able to work, at any time, users can have access to their personal data by relying on distributed data management solutions (e.g. Distributed file systems [1], Internet Workspaces, etc.) or by using mobile storage devices (e.g. USB flash drives, etc.) for maintaining replicas of their personal data.

The usability of distributed data management solutions is limited by the performance of network connectivity and the difficulty to setup a personal server for the average user in an environment full of private networks, firewalls, etc. The use of mobile storage devices tends to be more popular, but it force users to keep replicas manually synchronized and handle back-up copies to deal with the possible loss, theft or failure of the device. This problem is made worst since users tend to have multiple devices (both computers and mobile storage devices), with data replicas in each one. Users are also forced to deal with concurrent data updates whenever the latest version of their data is unavailable.

The goal of this work is to develop a system that allows users to always access their work or personal data, independently of their location (computer host) or machine connectivity, by combining the strengths of distributed data management systems and mobile storage devices. Replica management issues should be automatically dealt by the system, leaving users free from these tasks. Also, the system should allow concurrent access to replicas, thus automatically dealing with conflicting updates. The system should be able to use replicas stored both in mobile storage devices and remotely to provide the best service possible.

1.1 Proposed Solution

Mobile phones, not only make part of our current day's life, but also have considerable computing power, storage space and many different wireless communication capabilities. Our solution takes advantage of the storage capacity of these devices, for maintaining replicas of the users' data, and their wireless communication capabilities, for allowing these replicas to be accessed as regular files in any computer with a wireless communication interface. This way, users will always carry their personal files with them, and will be able to access them in any computer, independently of their location and connectivity.

The FEW Phone File System (FEW Phone FS) is a distributed data management system, relying on optimistic replication, that allows users to always access their personal data. It is based on the client-server model, using the mobile phone as a portable personal file server.

To address the storage limitations imposed by these devices, the FEW Phone FS allows for multimedia contents to be adapted to the device's specifications, thus reducing the volume of data stored on the mobile phone. This also contributes to the reduction of communications with the mobile phone, while reducing the device's energy consumption.

A data source verification mechanism allows the FEW Phone FS to keep track of the additional sources of files managed by the system. This mechanism allows the system to retrieve the URLs of the resources obtained from remote sites, thus allowing direct access to these resources whenever a high-speed network connection is available. When data stored in the mobile phones is an adapted one, this mechanism also allows users to access the full fidelity versions of the data, by obtaining original contents from other sources (or nodes in the system).

1.2 Contributions

FEW Phone FS presents a solution for data availability in a mobile computing environment, while also addressing the limitations imposed by such environments. The most important aspects of its design are:

– An optimistic replication mechanism, allowing for replicas to be created and accessed when needed, including a novel scalable update tracking solution. These characteristics allows for disconnected operation.

- Reconciliation techniques that automatically detect and resolve possible conflicting updates.
- A mechanism that allows for multimedia contents to be adapted to the specifications of mobile devices, reducing storage and communication overhead.
- A mechanism for identifying the replicas of the user's data, allowing access to any of the existing replicas based on that information.

2 Design

The FEW Phone FS is a distributed data management system based on the client-server model. The mobile phone (or multiple mobile phones) acts as a portable file server for the system, maintaining the user's data, while the computers act as clients, handling file system calls made to the system. The FEW Phone FS also allows peer-to-peer interaction among clients, thus addressing power limitations imposed by the mobile phone.

The system relies on an optimistic replication design that allows for long lived replicas to be created and accessed in any client (computer), at any time. This way, client requests can be fulfilled directly from the long lived replicas stored on the local file system, thus contributing for the reduction of the mobile phone's power consumption due to communications.

Data replicas are synchronized between the client and the server by a periodic reconciliation process, whenever the mobile phone is close to a computer. As we expect the user to always carry her mobile phone, this process guarantees that the most recent version of the user's data will be kept in the mobile phone. Thus, users will always have access to the most recent version of their work, even in disconnected machines. Additionally, the system guarantees that data replicas stored in computers are synchronized as users move close to them.

The optimistic replication approach also allows disconnected operation, even when the user forgets his mobile phone. This increases the possibility of conflicting updates to occur. The reconciliation process is designed to automatically detect and resolve conflicting updates, combining the "last version wins" rule and a "no update is lost" policy, as explained in section 2.2.

The FEW Phone FS operates on sets of files known as *containers*. A container is a subset of the file system tree. It can be seen as a file system that has its root on some base directory. The reconciliation process typically operates at the container granularity, although it is possible to synchronize only a subset of the container.

The use of a mobile device as a mobile phone, forces the FEW Phone FS to deal with the limitations imposed by this kind of devices.The FEW Phone FS addresses power consumption limitations by reducing the number of tasks preformed on the mobile phone, and by reducing both network interactions with the mobile device as well as the volume of data transferred during these interactions. Storage capacity limitations are addressed by reducing the volume of data written to the mobile phone's memory. The integration of a Data Transcoding (DTC) and a Data Source Verification (DSV) module are essential for accomplishing these goals.

The FEW Phone FS is designed to take advantage of the available wireless technologies. For this purpose, it features a modular communications component that allows users to add new connectivity modules enabling the system to be further extended. The server (mobile phone) is responsible for determining which technology to use for the communications channel. The choice is based on the available technologies and the power state of the mobile phone. The decision will select the technology that presents the best performance/power consumption ratio.

2.1 Version tracking mechanism

Each replica managed by the FEW Phone FS maintains a set of attributes (metadata). These attributes allow the system to address replicas unequivocally (i.e. by their globally unique identifiers (GUID)) and also to determine the current state of each replica.

Each replica keeps, for each file, a version vector [2] that maintains a summary of the updates performed on the file. A version vector V for a file f is maintained as a set of pairs (Si, Ci), where Ci represents the number of updates performed on f, at site Si. We define $V(Si) = Ci$ when $(Si, Ci) \in V$, and $V(Si) = 0$ when $(Si, \_) \notin V$.

Unlike the common approach [3, 1, 4, 5], the FEW Phone FS uses an 'updated' flag to mark replicas as updated instead of immediately updating the version vector. This is used to prevent adding new entries to a replica's version vector, by using existing entries to record updates whenever possible. For example, consider two synchronized replicas of file 'x', one in node 'A' and the other in node 'B', with a version vector with only one entry for node 'B'. Without loss of generality, suppose the version vector entry for B is 1, i.e., the version vector can be represented by the set $(B, 1)$. With our approach, an update to the replica in node 'A' will mark that replica as updated, instead of adding a new entry to the vector (the normal approach would be to update the version vector to $(A, 1), (B, 1)$). During reconciliation, the "updated" flag signals the update, and, since the version of both replicas is the same, the vector is incremented in the entry of 'B' instead of creating a new entry for 'A'. This is possible as version vectors are used to keep track of updates, making no difference where the updates are recorded (as long as there is no concurrent updates). If concurrent update exist or no replica involved in the synchronization has an entry in the version vector, new entries are created to record the updates, as usual.

This approach allows the system to minimize the number of entries in the version vectors, thus contributing for reducing space overhead, while improving scalability. In our scenario, where the synchronization process tends to include a single or a small number of mobile phones, it is usually possible to keep just a entry for each mobile phone in the version vectors, independently of the number of replicas created in other computers.

2.2 Reconciliation

The reconciliation process is divided into two stages: directory reconciliation and data reconciliation.

Directory reconciliation allows the system to reach a coherent directory structure from two, possible conflicting, structures. For this purpose, a log is used for storing directory update information. Each node maintains a replica of this log, adding new entries every time a directory update is executed. Each log entry contains an unique entry identifier, the type of update executed, the GUID and the version of the updated object. A log entry's unique identifier allows the system to, in case of conflict, reach a consistent state during the reconciliation process. Possible directory updates are: create, delete, and rename file/directory. A version vector [2] is also associated with the log structure, and is used to store a summary of the updates produced by each node to the directory structure.

During the directory reconciliation process the log's version vector is used for determining the unknown updates between nodes. Each node's unknown updates are then exchanged and executed respecting their causal order. The convergence of both replicas is guaranteed since we have designed operations such that every pair of concurrent operations commute (see [6] for more details).

The basic idea for making concurrent operations commutative is to allow different objects to have the same symbolic name, but different GUIDs. Thus, operations over different GUIDs trivially commute. For operations on the same GUID, we implement a *write with the later timestamp* wins policy. For objects with the same symbolic name, the system deterministically adds some disambiguator to one of the objects, while internally maintaining the original name. Users are free to keep the system as it is, but they are expected to resolve the conflict by renaming one of the objects.

The data reconciliation process allows the FEW Phone FS to synchronize the contents of the stored replicas. To optimize the reconciliation process, a file update marks all directories in the file's path as updated, all the way to the file system's root. This simplifies the data reconciliation process, since it guarantees that the contents of two replicas of the same directory are synchronized when neither one is marked as updated. This way, the reconciliation process can skip those directories. The 'updated' flag is also used to avoid the propagation of update information to already marked directories, allowing this process to stop when it detects an already marked directory.

Concurrent updates are resolved by selecting the replica that reflects the update with the largest identifier using the Lamport [7] order on pair (counter, site ID). This guarantees that the same version wins, independently of the number of replicas involved. The "loser" version is moved to a special "lost and found" directory, allowing users to restore it if needed.

3 Addressing Mobile Phone's Limitations

Using a mobile phone as a portable personal file server requires using extensive storage space and communications in the mobile phone. In this section, we describe two modules of the system that help addressing these issues.

3.1 Data Transcoding Module

Currently, users tend to have large amounts of high-quality multimedia data, such as digital photographs, music and videos. In many situations, the user can satisfactorily use lower-quality version of the data. For example, an 8 mega pixels digital photograph with 32 bit color depth can only be presented with a resolution of 320x240 pixels and a color depth of 16 bit, in most high-end phones, and a 24 bit depth is often more than satisfactory in any computer.

The Data Transcoding module (DTC) is designed to convert multimedia contents to best fit the specification of the mobile phone (or other specified by the user). In the previous example, the 8 mega pixel digital photograph can be rescaled and its color depth decreased in order to best "fit" the mobile phone's screen specifications, thus reducing its data size. This leads to less storage consumption while reducing the volume of data transferred to the mobile phone.

The DTC module is used during the reconciliation process, before transferring multimedia contents to the mobile phone. Re-encoded versions are transferred instead of the original ones. This process allows users to access these contents on the mobile phone or in other computers, reducing the storage and communications required to access them. When combined with the following module, it can also allow users to access the full-fidelity version in other computers.

3.2 Data Source Verification Module

Currently, an important fraction of the files stored by users have been obtained from remote sites - e.g. the web. Additionally, an increasing number of on-line storage services allow users to rely on these services for storing or sharing personal data. These systems can be used as alternative storage sources for users to access their data.

The Data Source Verification module (DSV) is designed to check and retrieve the source(s) of a file. If a file has been transferred from a remote site (e.g. HTTP), this module is used to obtain that site's URL. This way, a well connected machine (i.e. with a high-speed network connection) may use this information for retrieving data from the remote site, rather than from the mobile phone.

The DSV module works as a network proxy, by monitoring the user's connections. Whenever a file is updated in the local disk, the system verifies if it has been obtained from a remote site, by comparing a secure hash (SHA-1) of the stored file with recently transferred contents. If the new contents have been obtained from a remote site, the URL of that site and the secure hash of the contents are added to the replica's metadata. During the reconciliation process

this information is transferred to the mobile phone together with the replica's contents (or the transcoded version for multimedia contents). When the server is synchronizing with other sites and a high speed network connection is available, the URLs can be used to transfer those contents from the remote sites, instead of using the replica stored in the mobile phone. The secure hash stored in the metadata is used to verify that the URL still references to the same contents. This approach allows to reduce communications with the server, which also reduces power consumption.

FEW Phone FS nodes can also be used as data sources for retrieving data contents. This way, each file kept in the server also maintains the URLs of the sites that store a replica of that file. Availability improves with the number of replicas in the system. The number of references per replica can be used to determine the need for maintaining those contents in the mobile phone. In the limit, the server can be used for storing only the URLs (and other metadata) of the available replicas.

This module also allows the system to access the full-fidelity versions of the stored data. Thus, transcoded versions are only transferred from the server when the original version is inaccessible. In this scenario, the transcoded contents and the URLs for the original ones are transferred to the client, and the new replica is marked as 'not original'. Whenever the replica is accessed, the system will try to retrieve the original contents from the available sources. This process is repeated until the full-fidelity version is transferred to the node, at which time the replica is marked as 'original' and the URLs are discarded.

4 Evaluation

The current prototype of the system was developed in Java. The client side was developed for Linux, relying on FUSE-J for intercepting file system calls (FUSE-J is a Java wrapper for the FUSE [8] system), and was executed on a desktop computer with the specifications presented in Table 1. The server side was executed on an Apple iPod Touch 1G.

In this section, we present some performance results obtained using our prototype. The results presented next are always the average of 8 runs, after removing the highest and lowest obtained results during those runs. All read and write tests were preformed with a clean file system cache (after the system was rebooted).

4.1 Read-Write Evaluation

The results presented in Table 2 were obtained by compressing files into an archive, thus evaluating the read performance. The archive was stored in the local file system, and was created using the 'tar -czf' command. The source files were stored on the local file system and in the FEW Phone FS.

From the obtained results it is possible to observe the reduced and constant overhead imposed by the FEW Phone FS. The overhead does not affect the usability of the system for normal operation.

Table 1. Evaluation environment specifications.

Operating System:	Linux Ubuntu 8.04.2
Kernel version:	2.6.24-23
CPU:	Intel® Core™2 Duo T7200 @ 2.00 GHz
RAM:	2.00 GB
File System:	ext3
FUSE version:	2.7.2
FUSE-j version:	2.4 pre-release 1
Bluetooth:	v2.0 + EDR
Wi-Fi:	802.11g

Table 2. Measured times for adding files to an archive.

Total Files Size	**Number of Files**	Adding files from		**Overhead**
		Local FS	**FEW Phone FS**	
648 $kBytes$	84 files	0.193s	0.235s	$\approx$ 22%
2.3 $MBytes$	89 files	0.363s	0.447s	$\approx$ 23%
171 $MBytes$	1108 files	14.644s	18.068s	$\approx$ 23%

The results presented in Table 3 were obtained by decompressing packed archives into the local file system and into the FEW Phone FS, using the 'tar -xzf' command on different size packets. This experiment evaluates the performance of write operations.

Table 3. Measured times for extracting files from an archive.

Archive Size	**Number of Files**	**Unpacked Size**	Extracting files to		**Overhead**
			Local FS	**FEW Phone FS**	
100 $kBytes$	84 files	648 $kBytes$	0.082s	0.243s	$\approx$ 296%
495 $kBytes$	89 files	2.3 $MBytes$	0.110s	0.367s	$\approx$ 334%
151 $MBytes$	1108 files	171 $MBytes$	16.876s	25.590s	$\approx$ 151%

These results present a non-negligible overhead when compared to the values obtained from the extraction of the same contents to the local file system. This overhead is due to the management of information for reconciliation use, which is inefficient in our current implementation. Although the overhead is large, we believe that for common uses, this does not affect the usability of the system, as it is hardly noticeable for the user - for example, when writing 89 files with a total size of 495KB, the total execution time is still under 0,5 seconds.

4.2 Reconciliation Evaluation

The following results were obtained during the reconciliation process. The results presented in Table 4 were obtained during the reconciliation of two already synchronized nodes. In these results it is possible to see the low overhead of this process, thus showing the advantages of the update detection methods used.

Table 4. Measured reconciliation times for synchronized nodes.

PC to iPod using Wi-Fi
0.396s

The results presented in Table 5 were obtained during the reconciliation process, using two different set sizes, with a clean server side cache (first reconciliation). From these results it is possible to see the limitations imposed by the low bandwidth of Bluetooth technology.

Table 5. Measured first reconciliation times.

Number of Files	**Total Size**	**Reconciliation Time**		
		PC to PC		**PC to iPod**
		Bluetooth	Wi-Fi	Wi-Fi
84 files	648 $kBytes$	17.276s	6.876s	35.762s
89 files	2.3 $MBytes$	30.158s	8.856s	51.706s

The results presented in Table 6 were obtained during the reconciliation of a second PC with the iPod Touch, after the initial synchronization. These results present the improvement in performance when using the information retrieved by the DSV module. Using peer-to-peer interaction, not only improves system performance, but also reduces the volume of data transferred from the server.

Table 6. Measured second reconciliation times.

Number of Files	**Total Size**	**Reconciliation Time**		**Gain**
		with Peer-to-Peer	without Peer-to-Peer	
84 files	648 $kBytes$	14.877s	16.921s	$\approx$ 14%
89 files	2.3 $MBytes$	26.338s	30.393s	$\approx$ 15%

The results obtained during the reconciliation of multimedia contents are presented in Table 7. These results were obtained running the server with a clean cache, and with a client storing two 15 mega pixels digital photographs, with

approximately 4MBytes each. The two photos were transcoded to a resolution of 800x533, that resulted in a reduction in size to approximately 33kBytes.

Table 7. Measured multimedia reconciliation times.

Number of Files	**Total Size**	**Reconciliation Time**		**Gain**
		with Transcoding	without Transcoding	
2 files	8 $MBytes$	8.549s	27.761s	$\approx$ 325%

From the obtained evaluation results, it is possible to conclude that the use of both the Data Transcoding and the Data Source Verification modules has proved to be beneficial for the performance of the system, improving the system's performance, and also reducing communications and storage overhead.

5 Related Work

A large number of distributed file systems have been implemented (e.g. Ficus [5], Coda [1]), allowing users to remotely access their files. Some of these systems can be used in mobile computing environments and include support for disconnected operation. However, the complexity associated with setting up a new server and using it in a network environment with private networks and firewalls lead most users to prefer using portable storage devices to transport their data.

Lookaside caching's [9] allows for portable storage devices (PSD) to be used as availability extensions and performance enhancers for distributed file systems (DFS). Although Lookaside caching allows for disconnected operations, users can only access a small portion of their data during disconnection, as the portable storage device is used as a cache. Additionally, this approach still requires the use of a distributed file server, incurring in the same drawbacks of distributed file systems.

PersonalRAID [10] allows a PSD to be used for propagating updates among several personal replicas. However, as the system only maintains updates in PSDs, makes it impossible for a user to access all his data in a new computer.

Footloose [11] introduces the concept of physical eventual consistency. This concept allows for the mobile phone to be used as the means to propagate updates between disconnected nodes. This is possible since the mobile phone stores the most recent version of the user's data, and it is "carried around" by the user. FEW Phone FS extends the approach of Footloose by allowing multimedia data to be transported efficiently and by allowing clients to obtain data contents from other replicas (even outside the system), thus minimizing the requirements of the mobile phone.

Like Coda [1], the FEW Phone FS also uses a log structure for maintaining directory updates. Although both systems automatically deal with directory update conflicts, the solution used by Coda relies greatly on servers. This solution

is not suitable for the FEW Phone FS due to the limitations of mobile devices. Coda deals with file update conflicts using application specific resolvers. Other system, like Pangaea [3] also use this approach. This approach could also be used in the FEW Phone FS.

EnsemBlue [12] is a distributed file system for personal multimedia that incorporates both general-purpose computers and consumer electronic devices. The system transcodes data, based on application needs, using what the authors describe as 'persistent queries'. In the FEW Phone FS the Data Transcoding module is used during the reconciliation process for adapting multimedia contents based on the specifications of the mobile device.

quFiles [13] provides a mechanism to support multiple fidelities of data within the same file system. The transcoding process, in a normal scenario, only happens once for each file. quFiles transcodes files when they are stored in the system, maintaining multiple transcoded versions of each file. The FEW Phone FS transcodes files "on-the-fly" during the first synchronization with the mobile phone, since these are transcoded according to the specific device's specifications.

6 Conclusions

The main goal of this work was to design and develop a distributed data management system, that would allow users to access their personal data in any machine, independently of location and network connectivity.

For this purpose, the FEW Phone File System was designed to take advantage of current mobile phones' storage and wireless communication capabilities for maintaining the most up-to-date version of the users data, and allowing for replicas to be created and accessed whenever needed.

The combination of the client/server model with peer-to-peer interaction leads to an hybrid architecture. Relying on the mobile phone as a portable server allows the system to use it to provide data availability at any time. Peer-to-peer interaction allows for power consumption to be reduced and for improving the system's performance, as the presented results have shown.

The Data Transcoding module allows to reduce multimedia data transferred to and stored on the mobile phone, without compromising quality when these files are accessed in the mobile phone. This can be interesting for example, for digital photographs that a user wants to keep in both the mobile phone and his desktop computers.

The Data Source Verification module allows the system to record the sources for the files stored in the system, including remote sources (not in the system). This approach explores the common case where files stored were obtained from the Web or are stored in some remote server (e.g. CVS/SVN). This module also stores the location of other replicas in the system. This information is used by clients to obtain replica contents from other clients. To our knowledge, the FEW Phone File System is the first system to use this approach to improve performance and availability. Moreover, this also extends battery life by reducing communications with the mobile device, while improving performance.

By combining the Data Source Verification module with the Data Transcoding module, the system allows clients to always access full-fidelity contents. This is a new feature when compared with previous solutions that used data transcoding.

The results of the performance tests showed that the proposed design imposes minimal overhead to data access on the clients, after initial synchronization. Also, the use of the Data Transcoding and Data Source Verification modules allow the FEW Phone File System to achieve its goals.

References

[1] Braam, P.J.: The Coda Distributed File System. Linux J. (1998)
[2] Parker, D.S., Popek, G.J., Rudisin, G., Stoughton, A., Walker, B.J., Walton, E., Chow, J.M., Edwards, D., Kiser, S., Kline, C.: Detection of Mutual Inconsistency in Distributed Systems. IEEE Trans. Softw. Eng. **9**(3) (1983) 240–247
[3] Saito, Y., Karamanolis, C., Karlsson, M., Mahalingam, M.: Taming aggressive replication in the Pangaea wide-area file system. In: Proc. OSDI. (2002) 15–30
[4] Guy, R.G., Reiher, P.L., Ratner, D., Gunter, M., Ma, W., Popek, G.J.: Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication. In: ER '98: Proceedings of the Workshops on Data Warehousing and Data Mining, London, UK, Springer-Verlag (1999) 254–265
[5] Guy, R.G., Heidemann, J.S., Mak, W., Popek, G.J., Rothmeier, D.: Implementation of the Ficus Replicated File System. In: USENIX Conference Proceedings. (1990) 63–71
[6] João Soares: FEW Phone File System. Master's thesis, Faculdade de Ciências e Tecnologia, Portugal (April 2009)
[7] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7) (1978) 558–565
[8] Henk, C., Szeredi, M., Pavlinusic, D., Dawe, R., Delafond, S.: Filesystem in Userspace (FUSE) (December 2008) http://fuse.sourceforge.net/.
[9] Tolia, N., Kozuch, M., Satyanarayanan, M.: Integrating portable and distributed storage. In: Proceedings of the 3rd USENIX Conference on File and Storage Technologies. (2004) 227–238
[10] Sobti, S., Garg, N., Zhang, C., Yu, X., R, A.K., Wang, O.Y.: PersonalRAID: Mobile Storage for Distributed and Disconnected Computers. In: Proc. First Conference on File and Storage Technologies. (2002) 159–174
[11] Paluska, J., Saff, D., Yeh, T., Chen, K.: Footloose: a case for physical eventual consistency and selective conflict resolution. Mobile Computing Systems and Applications, 2003. Proceedings. Fifth IEEE Workshop on (Oct. 2003) 170–179
[12] Peek, D., Flinn, J.: EnsemBlue: Integrating distributed storage and consumer electronics. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation. ACM SIGOPS. (2006) 219–232
[13] Veeraraghavan, K., Nightingale, E.B., Flinn, J., Noble, B.: quFiles: a unifying abstraction for mobile data management. In: HotMobile '08: Proceedings of the 9th workshop on Mobile computing systems and applications, New York, NY, USA, ACM (2008) 65–68

Wiinteraction: A study on smart spaces interaction using theWiimote

Hugo Seixas, Nuno Salgado, Rui José

University of Minho, Portugal

Abstract. This paper describes a study on the use of the Wiimote as a generic interaction device for smart spaces. We have identified the range of interaction possibilities that can be explored when creating Wii-based interfaces for smart spaces and we have explored the application of those interactive features in the context of a typical museum guide. While there are many features that can be explored creatively to sustain the use of the Wiimote as a generic interaction device, we also found that there is at least one critical requirement that is not supported. More specifically, we have identified the need to include access to rich information, such as the one provided by digital displays. We thus propose a shared control mechanism for public displays that enables a user equipped with a Wiimote to first gain control and then browse information in a public display. The results of our study show no major limitation in the proposed approach, but identify device discovery as one major technical flaw that still needs to be overcome before the wiimotemay realistically become a generic interaction alternative for smart spaces.

Keywords: Wiimote, interaction, public displays, museum guides

1 Introduction

Exploring new interaction models has always been a very active research topic in ubiquitous computing, the overall goal being to enable interaction with computers in the multiple situations of everyday life in which the traditional desktop metaphors no longer apply. Small mobile devices, such as mobile phones and PDAs, have been extensively explored for this purpose, but in their essence they are still very close to the desktop metaphor. Being designed as universal tools they are too complex and distracting for many scenarios in which physical exploration is the core of the user experience. On a completely different path, there has also been considerable research in conceiving and experimenting with multiple types of dedicated sensor and interaction devices, but to date we still do not have any alternatives with widespread use and easy deployment.

In this work we explore the potential of the Wii Remote as an alternative interaction device for situated services. The Wii Remote, which has unofficially been nicknamed "Wiimote", is the wireless controller for Nintendo's Wii console. The Wiimote is equipped with accelerometers, enabling motion sensing and gesture recognition, an optical sensor (Infrared), numerous buttons and also basic output through four blue LED lights, a rumble and a speaker. In addition to its technical features, there are some important points that can make it very attractive as an interaction device for smart spaces scenarios: firstly, the Wiimote has been reversed engineered and there are now numerous libraries enabling it to be used as generic

controller for any computer and using multiple programming languages; secondly, Wiimotes are an off-the-shelf product that is robust, widely available, and can be bought separately at a price range that is comparable to the price of a wireless mouse; thirdly, with the expansion of the game console market, these devices are increasingly part of the digital culture, with many people being already familiar with its commands and interaction modalities. This means that many people will be able to achieve a reasonable level of intuitive and easy handling, without the need for any long, descriptive learning processes.

Despite this obvious potential, the Wiimoteoriginates from the game consoles world and was conceived for a rather specific scenario, being far from obvious how its features can be leveraged as enablers for interaction in smart spaces.In this work, we take the almost canonicalexample of the museum guide as the background for a study on the interaction design space of the Wiimote. Our purpose with this study is to explore the range of alternatives for Wiimote-based interactions and provide a framework that can be useful to smart space designers. We have also implemented a shared control mechanism that enables Wiimotes to gain access and control a public display for information access. This has enabled us to test the overall technical feasibility of the proposed system and to evaluate some of the interaction concepts with users.

2 RelatedWork

There has been considerable work in exploringnew ways to use Wiimotes through some type of hacking.The most well-known and influent work has been done by John Chung Lee who published numerous projects of new applications for the Wiimote:

- Tracking Your Fingers with the Wiimote. "*Using an LED array and some reflective tape, you can use the infrared camera in the Wii remote to track objects, like your fingers, in 2D space."* [1]
- Low-Cost Multi-point Interactive Whiteboards Using the Wiimote. "*Since the Wiimote can track sources of infrared (IR) light, you can track pens that have an IR led in the tip*." [1]
- Head Tracking for Desktop VR Displays using the WiiRemote. "*Using the infrared camera in the Wii remote and a head mounted sensor bar (two IR LEDs), you can accurately track the location of your head and render view dependent images on the screen.*" [1]

There are many other similar projects, and in particular many projects focusing on the use of the accelerometers for gesture recognition:

- Gesture Recognition with a Wii Controller is a project that studied "*recognizing gestures to interact with an application and present the design and evaluation of our sensor-based gesture recognition*". [2]
- Wii Remotes as Tangible Exertion Interfaces for Exploring Action-Representation Relationships is a project that based their study in: "*Sensor technologies and exertion interfaces offer new opportunities for interaction with digital data. Technologies such as Wii Remotes (…) "*. [3]

- Wave Like an Egyptian — Accelerometer Based Gesture Recognition for Culture Specific Interactions. The authors of this project studied "*how the Wiimote can be utilized to uncover the user's cultural background by analyzing his patterns of gestural expressivity in a model based on cultural dimensions*". [4]

Another important type of work regarding the Wiimote, are the Wiimote libraries, through which Wiimotes can be integrated as interaction devices for any computer program.

- WiiGLE - Wii-based Gesture Learning Environment is the implemented project that originates the paper referenced before. [5]
- WiimoteLib: It is written in C# and VB.NET and consists on using nintendo controllers in .NET Applications. [6]
- WiiUse: It is a library written in C that supports all features except gesture recognition. [7]
- WiiUseJ: It is a Java API to use Wiimote on the computer. Is bult on top of WiiUse.[8]
- WiiRemoteJ: It is another Java API to use Wiimote on the computer. [9]
- Wiigee: Implemented in Java, "is an open-source gesture recognition library for accelerometer-based gestures specifically developed for the Nintendo® Wii™ remote controller." [10]
- GlovePIE: Is the Glove Programmable Input Emulator. It can be understood like a script interpreter that allows a basic mapping by Wiimote (or other joystick device) to keyboard keys. [11]

For a more detailed list of projects and Wiimote based documentation, consult [12], [13], [14] and [15].

Our work takes a different approach, in that we are not trying to propose any new sophisticated Wiimote-based interaction, but instead trying to understand how the Wiimote could be leveraged as a generic, off-the-shelf device for mundane interaction in everyday life.

3 Exploring the interaction design space

The Wiimote provides a wide range of sensing and actuation mechanisms that were originally conceived to integrate game experiences. Our first task in studying the role of the Wiimote as a tool for smart spaces was to study those multiple interaction mechanisms from the perspective of their most basic affordances. The objective was to clarify the range of possibilities that can be explored when creating wii-based interfaces for smart spaces. We have identified the following categories of sensing and actuation mechanisms:

- **Detecting a Wiimote** – The most basic function of the Wiimote is as a presence detection device. Given the use of Bluetooth as the connection technology, a simple process of Bluetooth scanning is enough to obtain information on which Wiimotes are currently near each Bluetooth access point. This can be done even if the Wiimote is not connected with the scanning device. In a scenario of one

Wiimoteper user, this may be used as a process for detecting the presence of a particular person.

- **Basic Button Interaction**–The Wiimote is equipped with a diverse set of buttons. Any arbitrary action can be triggered either by pressing a single button or by using some preset combination. As long as some meaning is communicated to the use of those buttons they can easily be associated with any actions.

- **Accelerometers** – The accelerometers offer great potential as an interaction enabler, both for implicit and explicit forms of interaction. Based on the level of involvement required by users, we have identified the following main types of use for the accelerometers:

    1. Basic activity recognition: This corresponds to using accelerometer data for recognising possible activities by the owner, such as walking or seating. While the interpretation of the data generated may not be very robust in general, the interpretation within the context of a specific place and within the framework of the activities common to that place, may offer a simple and yet useful mechanism for basic activity recognition.
    2. Explicit actions through basic movements: This corresponds to basic movements that need to be communicated, but do not require any training. This may include simple gestures such as swinging the Wiimote, gently hitting with the Wiimote on a particular surface, or bumping two Wiimotes together (for example as a pairing mechanism). The main idea is that gestures should be simple enough not to require any training.
    3. Trained Gestures for Interaction: If we consider that people will have the opportunity to train more elaborate gestures, we can use many more types of movements as interactive actions. For example, drawing a square or a circle with the Wiimote to trigger some specific action. If the person trains the gestures, it is also possible to train "secret gestures" defined by each user, which can serve as signatures or authentication signals. That kind of signatures or authentications could be some mechanism of identification for the users in the scenario of shared Wiimotes. However, using trained gestures just to trigger a particular action may not be very efficient because it requires training, it is prone to errors, and it may be awkward for someone in a public place. This type of gestures is probably more suitable in scenarios similar to games in which the intensity and extension of the gestures is itself relevant.

- **IR LEDs in a grid seen by the Wiimote camera**–Given that the infrared camera in the Wiimote can very accurately detect the position of points of infrared light, it becomes simple to use infrared LEDs in a grid in order to allow pointing, moving or dragging actions. This corresponds to the standard approach used by many Wii games in which the distance and inclination of the Wiimoteis determined by measuring the relative position and relative distance of light points generated by the LEDs in a standard grid with known points placed at known distances. This feature can be understood as the movement of a mouse in a

standard desktop system. Alternatively, it should also be possible to have multiple grid patterns and use them to recognise specific locations associated with those patterns.

- **IR pointed to LEDs** - This feature is dual to the one stated above. In this scenario, Wiimotes are static and actively detecting the movement of the LEDs, which will be the ones performing actions. In our scenario, this is not applicable because we are considering that people are carrying the Wiimote.

- **Output** - Wiimotes come with three different ways of providing output:

    1. Audio: a built-in mono speaker can produce beeps to notify the user of any relevant events or some very relevant system state.
    2. Lights: The four LEDS can be used in combination to communicate multiple system states or alerting the user for some specific system state.
    3. Rumble: a small vibrating engine is incorporated in the device and can be used for notification or as simple feedback mechanism to communicate that a particular action has been correctly received, understood and executed.

    These three techniques are complimentary in providing output to the user. The audio and the rumble can be especially useful when there is a need to notify the user of some event. Pre-determined sounds our rumbles can be associated with a short number of relevant events, such as entering a new interaction area or being in proximity of something relevant. The lights are more useful to represent pre-defined states, allowing the user to glance them at any moment to get a quick update on the state of some system variable. This can also be combined with buttons to represent on request the state of specific variables.

4 A Wiimote-based museum visit

This work did not include any real-world deployment in which we could get more information on the practicalities of deploying and using the Wiimote in smart spaces. Instead, we looked at some common digital museum guides and studied how we could support similar functionality using the features available in the Wiimote. We specifically chose this very common scenario for smart space interaction because we thought it would make it easier to understand some of the base requirements and compare some of the results. Building on the interaction space outlined in the previous section, we have made a series of brainstorm sessions on how the Wiimote could be used as the basis for an interactive visit to a museum.For a complete understanding of a typical scenario we tried to find a story that could explain our intentions. We arrived at the following scenario:

Saturday. It's still very early in the morning, but family Molyneux is already in a hurry.

"Culture and Arts: Picappo paintings on exposure at Louvle Museum this weekend". The advertisement flooded newspapers and media the past days and lots of people are expected to come, so they want to arrive soon and avoid an endless waiting line at the entrance.

Louvle is huge. A complete tour would need at last two days to fully visit everything. Albert and Elvire, both art teachers at University, are happy for the opportunity to see one of the best art collections in the world. At the box office, the Molyneux are given a Wiimote that will accompany them during the whole day.

The twins, Enzo and Angeline, however, are not into boring cubism paintings and insist to go and see the science and technology interactive exhibit, on another wing of the Louvle, so another Wiimote is requested. Before departing, the family approaches a terminal to pair both devices. Now the museum system knows that these two Wiimotes are related to each other.

It's the first time at the museum, so they don't know their way around. Their Wiimote rumbles, warning them that there is interactive content nearby. They approach the terminal and look for information. A map shows the way for Picappo Gallery and they go straight there. All their movements are being tracked through the Wiimote and their interactive actions stored: now the museum system knows that the way to Picappo Gallery was searched for and it is the objective of those people.

Finally they reach the gallery. In the room, there are multiple displays where, by using the wiinteraction system, one can find more detailed information on the paintings. Albert and Elvire watched carefully and thrillingly all the masterpieces. The museum knows they didn't leave the gallery for the next three hours.

There was a painting that both loved particularly, so Albert "selects" it with the Wiimote and it is automatically sent to the souvenir's shop where a replica will be printed. At the end of the day, they can go pick it up and bring it home, where they will hang it by the fire pit.

As the lunch time approaches, they go to the restaurant zone. At a terminal they "ring" the other Wiimote, as they were previous paired. Now, as soon as the other Wiimote enters a Wiimote spot (there are hundreds over the museum) it will rumble and give a sonorous warning to their children. By consulting a map, the kids know where their parents' Wiimote is, and head up to the meeting place.

From a conceptual perspective, the use of the Wiimote as the interaction device for this scenario does not seem to offer any major problem. Depending on the back-office functionality, people would be able to trigger any arbitrary actions, coordinate their visit in a group, receive notifications, or bookmark any exhibit for future reference. The only crucial exception was rich information output, such as the one that can be obtained in the screen of a PDA or in a public display.

One of the most common scenarios in any sort of digital guide is the possibility to obtain more information about exhibited items. This is not naturally supported by the Wiimote, which does not have any type of screen. To overcome this limitation, we assumed that the museum would occasionally have some sort of public display and that the Wiimote would be used as an interaction device for those displays.Our implementation is focused on this particular feature and our study specifically addresses to what extent this specific use of the Wiimote could be efficiently supported. Ourscenario is based essentially in basic button interaction and also the output features, coupled with the use of the Wiimote for interaction with public displays.

4.1Implementation

Our implementation is focused on addressing the specific scenario of a public display in a room in a museum that is available for interaction by holders of Wiimotes. As represented in Fig. 1, the system consists in a set of displays (WiiSpots – computer display with a bluetooth dongle) with which Wiimotes are able to connect.

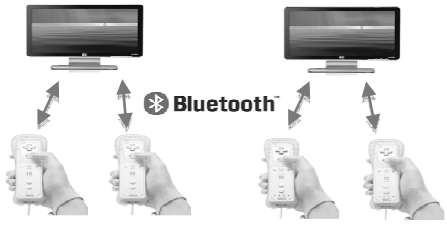


Figure 1: Wiimote interaction with public displays

The system can be seen by two distinct points of view. First, we have the back office that handles the access to the display, following a set of rules that define who has permission to use the application.Secondly, we have the system interface that allows users interaction. This is a web-based interface and should be independent of the implemented back-office. For this project we chose to use theWiiUseJ [14] Wiimote libraries, mainly because we wanted to implement the system in Java. We built an interface based on common menu navigation by binding the directional buttons on the Wiimote to the respective arrow keys that can be used in common menu navigation, as shown in Fig.2.

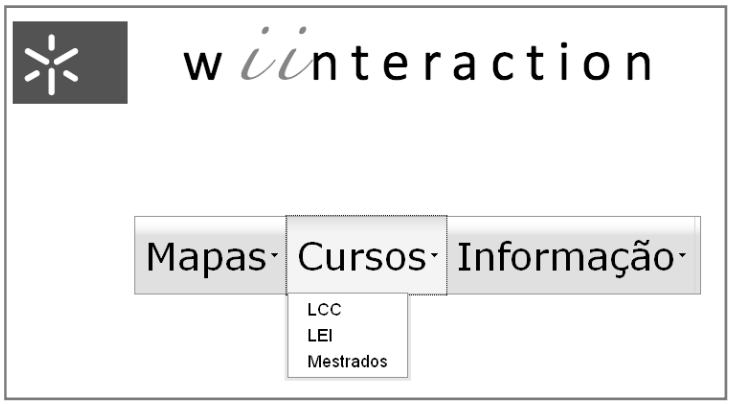


Figure 2: The winteraction display interface.

The front-end of our application is a common web-browser interface. We use the typical HTML and CSS to make the final proposal. The main menu is constructed using Yahoo UI Library. [20]. The most important reason to use Yahoo UI Libraries that this Library allows keyboard navigation. So, the idea is mapping keyboard keys into Wiimote buttons and use this solution to navigate the interface. This enabled the user to easily browse the menus as if they were using the arrows of a keyboard. This kind of solution makes our solution interface independent.

Given our multi-user scenario, we also needed to find a method not to allow a second user from interfering with the one "in charge" of the interaction at one particular moment. It was necessary then, to come up with some sort of protocol able to prevent that from happening, but at the same time, giving some fairness to the usage and avoiding monopolization of the display. So, as soon as a second user approaches a terminal already in use, he or she automatically enters a queue, giving the current user a maximum of three minutes to finish his activities, automatically giving the control to the second in the queue after that time. As soon as a new user is given priority, his Wiimote will rumble, giving him or her a light warning. A visual status of the queue is shown at all times in the screen. In the Wiimote that is currently in charge the left LED is also lightened up to signal control. The application of this access protocol is governed by the state diagram represented in Fig.3.

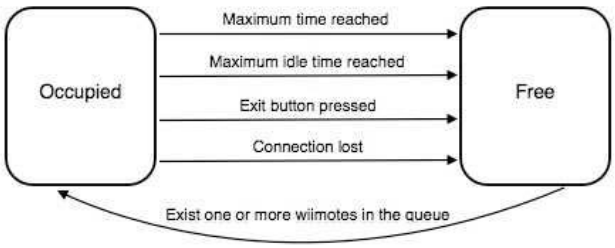


Figure 3: Transactions between display states.

The system also maintains a state for each Wiimote, which is represented in Fig.4

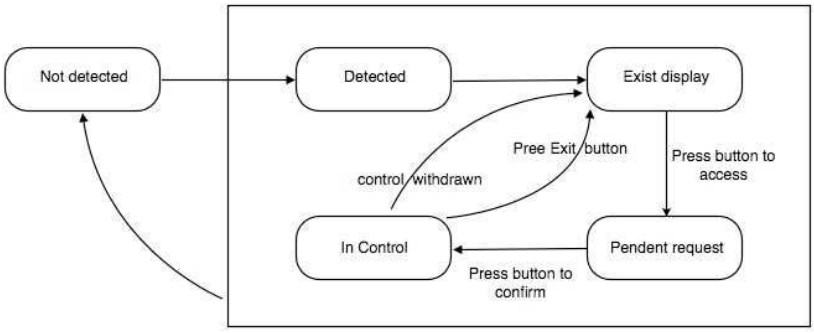


Figure 4: State diagram representing Wiimote states.

The initial state is the case where a particular Wiimote is not being detected. When it is detected, it changes to state detected, and if a public display exists it again changes to Exist Display. In this case, the Wiimotewill rumble and turn on one the LEDs. The user can now press a button to indicate the intention of using the display, causing another shift of the state. Another light will be on indicating the specific transition of the state. At this moment, the user is in the queue waiting for his turn.When his turn arrives, all the lights will blink repeatedly and the rumble will be activated once indicating the shift of the state. At this moment, the user is in control of the display.

Once in control, that control can be withdrawn, for example by exceeding the time limit, or, by pressing the exit button, with which the user can indicate his intention of dropping the control. The user also looses the control after a period of inactivity.

5 Results

To gain a better understanding of the feasibility of the proposed system, we created an implementation of the display control software and set an experimental deployment in which we could conduct some experiences with users. In this section we will now described our main results.

5.1Lessons learned from the implementation

A major effort in this work was devoted to dealing with the Wiimote libraries, and especially with the discovery process that needs to take place when Wiimotes are first detected. For example, using WiiUseJ, discovery is not possible and the only way to connect with Wiimotes is paring the device with wincom windows stack. We tried to use WiiRemoteJ for development. However, although the specification indicates that discovery is possible, the reality has been very different. First, the information related to this API is very limited. Secondly, even if we found information to solve some problems, the reality shows that very often it was not reliable.

Wiimote discovery using WiiRemoteJ seems to be possible in Windows, however, it is necessary to use bluecove [16] for discovery and bluesoleil [17] or widcomm [18] drivers switching the traditional wincom drivers. The problem seems to be solved, but again, we concluded that bluesoleil does not work under windows platform. We also find a big number of references for this in web forums. [19]

So, the last opportunity to solve the problem was to use bluecove with widcomm drivers. One more time we found problems. Widcomm drivers are a Broadcom solution for bluetooth devices and the problem is that these drivers do not support many number of bluetooth devices. In fact, none of our bluetooth devices was supported by this driver, so, we hadn't opportunity to conclude if it works or not.

Another important conclusion is related with the number of Wiimotes connected at the same time. Using WiiUseJ, by API limitation, the limit is four Wiimotes. However, if we shift our code to WiiRemoteJ API (hoping that the discovery problems will be solved), theoretically, there isn't any limit to the simultaneous

Wiimote connections. But, in practice, the Windows bluetooth stack imposes a limit for bluetooth devices connected at the same time, and that should be the limit.

In the interface we also found a technological limitation. Using yahoo UI we need to have the menu selected and clicked by the mouse to begin the navigation. There isn't any kind of solution for this problem in this library because, it is impossible to make the menu selected in its initial state. One possible solution is to use one button of the Wiimote as a mouse click and informing the user to click that specific button. The strategy is having the mouse pointer under the menu surface and after that click the menu should be able to be browsed. We solve this problem using Robot solution of J2SE API [21]. The tests indicate that this technical insufficiency is the major usability problem. Once in the application, almost all the users forgot that every time they enter the home page they needed to "click" in the mouse (button plus of Wiimote).

5.2 Users Tests

The target audience for this kind of solution is typically visitors of public spaces. In our specific case, we pretend to reach the visitor of DI (Informatics Department). Thus, we tested our system with five different people. In each case we explain to the person in what consist and what was the purpose of our system. We tried to explain that it would be important to the experience if they imagine that they were actually visiting a museum, or, in our particular case, imagine that they are visiting our informatics department and that there would be several WiiSpots around. We explain all the crucial issues and give to them a guide with all necessary explanations. We then gave them a set of tasks related with specific information available in the system and let them explore the system. At the end, we conducted structured interviews with the participants to collect information about their experience with the system.

The results of our trials have been fairly satisfactory regarding the response of the Wiimote usage by the people, even when our interaction environment is a simple test web page. We believe that one of the major factors of its success is the natural bond and amusement people automatically feel when given a Wiimote to their hands: all users showed immediate mastery at using the device. The management protocol for multiple users was also well accepted and thought fair.

Regarding the interface, the process was very simple to implement: we simply bound the buttons of the Wiimote to specific keys of a keyboard, allowing the user to intuitively perform his actions, demonstrated by the fact that no user needed an explanation on handling the Wiimote and interacting with the system during the trials. We noticed, however, that users had problems with the mouse click. In fact, every time the web page is accessed, users need to press the plus button of the Wiimote to select the proper menu (solution for the technical problem already described in the previous section) in order to map the mouse click, and thus, navigate the menu. Although we explain this question in the guideline, usually the users had problems with this process.

Up until this point, we always tested our system with a Wiimote already paired with the computer where the system was running. In one of our series of trials, we used a second Wiimote not yet paired with the system, confirming one of our

suspects: our libraries do not support active Wiimote discovery, working only with the Wiimotes paired at boot time. With this technical limitation, we can only manage up to a maximum of four Wiimotes at the same time, due to a restriction on windows Bluetooth stack. We hope that in the future, some kind of discovery service may be developed, that would allow us to fully develop our system features. Some testers mentioned the lack of information in the interface itself.

It was also obvious from several interviews that many people were expecting different interaction types. More than half the participants expressed that gesture recognition should be a better solution for this kind of application than basic button interaction. Even though they may be much less reliable, as was also pointed by other participants, gestures may have some advantages: The first is facilitating interaction without forcing the user to look at the Wiimote. Even though this can also be partially achieved with buttons, it may be much simpler by tilting the Wiimote. Also, using gestures may correspond more naturally to the user expectation of using the Wiimote, and that is also how we interpret this result. Even though our sample is small, this is an important conclusion to retain.

6 Conclusions

In conclusion, a lot of features are yet to be added in order to increase our system usability, particularly, taking advantage of the gesture capabilities of the Wiimote. Still, from our results we found nothing absolutely fundamental to makes us think that this idea may not be viable. There are still, some technical issues regarding discovery that need to be solved before this type of system may be put into practice in realistic scenarios. Despite our research, we were not able to find any service capable of supporting Wiimote Bluetooth discovery in a way that matched the requirements posed by our scenario.

Regarding the interactive possibilities, we consider that a solution based on gestures recognition should be an important alternative to basic buttons directions and that given widespread use, a set of conventions for basic commands could emerge that would cover most common scenarios.

References

1. Johnny Chung Lee website: http://johnnylee.net/projects/wii/
2. Schlomer, T., Poppinga, B., Henze, N., and Boll, S. - Gesture Recognition with a Wii Controller. A multi-university project of University of Oldenburg and OFFIS - Institute for Information Technology
3. Sheridan, J. G., Price, S. and Pontual-Falcao, T. - Wii Remotes as Tangible Exertion Interfaces for Exploring Action-Representation Relationships
4. Rehm, M., Bee, N. and André, E. - Wave Like an Egyptian — Accelerometer Based Gesture Recognition for Culture Specific Interactions. (2007)
5. WiiGLE Project - http://mm-werkstatt.informatik.uni-augsburg.de/project_details.php?id=46
6. WiimoteLib - http://www.wiili.org/index.php/WiimoteLib

7. WiiUse - http://www.wiiuse.net/
8. WiiUseJ - http://code.google.com/p/wiiusej/Sadsad
9. WiiRemoteJ - http://www.wiili.org/WiiremoteJ
10. Wiigee - http://www.wiigee.org/
11. GlovePIE - http://www.wiili.org/GlovePIE
12. Wiimote Project - http://www.Wiimoteproject.com/
13. WiiForPc - http://wiiforpc.com/index.php?page_id=3
14. WiiBrew - http://www.wiibrew.org/wiki/Index.php
15. WiiLi - http://www.wiili.org/index.php/Wiimote
16. Bluecove - http://www.bluecove.org/
17. Bluesoleil - http://www.bluesoleil.com/
18. Widcomm - http://www.broadcom.com/
19. WiiRemoteJ Forums - http://www.wiili.org/forum/
20. Yahoo UI Library - http://developer.yahoo.com/yui/
21. J2SE API - http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Robot.html

Sessão 4A: XML: Aplicações e Tecnologias Associadas (XATA)

$\mathcal{X}$AGra - An XML dialect for Attribute Grammars

Nuno Oliveira[1], Pedro Rangel Henriques[1], Daniela da Cruz[1], and Maria João Varanda Pereira[2]

[1] University of Minho - Department of Computer Science,
Campus de Gualtar, 4715-057, Braga, Portugal
{nunooliveira,prh,danieladacruz}@di.uminho.pt
[2] Polytechnic Institute of Bragança
Campus de Sta. Apolónia, Apartado 134 - 5301-857, Bragança, Portugal
mjoao@ipb.pt

Abstract. Attribute Grammars (AG) are a powerful and well-known formalism used to create language processors. The meta-language used to write an AG (to specify a language and its processor) depends on the compiler generator tool chosen. This fact can be an handicap when it is necessary to share or transfer information between language-based systems; this is, we face an interchangeability problem, if we want to reuse the same language specification (the AG) on another development environment.
To overcome this interoperability flaw, we present in this paper $\mathcal{X}$AGra - an XML dialect to describe attribute grammars. $\mathcal{X}$AGra was precisely conceived aiming at adapting the output of a visual attribute grammar editor (named Visual$\mathcal{LISA}$) to any compiler generator tool.
Based on the formal definition of Attribute Grammar and on the usual requirements for the generation of a language processor, $\mathcal{X}$AGra schema is divided into five main fragments: symbol declarations, attribute declarations, semantic productions (including attribute evaluation rules, contextual conditions, and translation rules), import, and auxiliary functions definitions. In the paper we present those components, but the focus will be on the systematic way we followed to design the XML schema based on the formal definition of AG.
To strength the usefulness of $\mathcal{X}$AGra as a universal AG specification, we show at a glance $\mathcal{X}$AGraAl, a tool taking as input an AG written in $\mathcal{X}$AGra, is a *Grammar Analyzer and Transformation system* that computes dependencies among symbols, various metrics, slices and rebuilds the grammar.

1 Introduction

In the area of language processing it is common to use *Attribute Grammars* (AG) [1] as a formalism to specify the language syntax, semantics and also the translation to target code. The notation of this kind of formalism strongly depends on the compiler generator tool (CG) we choose to automatically produce the processor. This implies the conversion of notations, in order to readapt the specification to other tools, which is a situation that occurs often.
The focus of this paper is on an XML dialect to represent universally attribute grammars, this is, in a tool-independent manner. The main idea is to generalize the output of AG

editing tools; instead of generating a description for a specific compiler generator, the editor-like tool under development can produce this general purpose dialect. Then to use this editor as a *Front End* (FE) for a specific generator, it is only necessary to resort a simple translatorto convert the XML description into the specific meta-grammar of that CG. This approach raises the usefulness of the editor-like tool, as it can be used as a FE for a larger range of grammar-based generators.

Actually, $\mathcal{X}$AGra — *an XML dialect for Attribute Grammars* — appeared during the development of Visual$\mathcal{LISA}$ [2], a visual editor for $\mathcal{LISA}$ [3] attribute grammars that enables the drawing of syntax trees (grammar productions) decorated with attributes and the respective evaluation rules to define visually a complete attribute grammar. In its original version, Visual$\mathcal{LISA}$ converts this visual description into $\mathcal{LISA}$ meta-language. To extend the use of that visual environment for AG specification, in order to cope with different compiler generator toolswas built to translate the visual representation into $\mathcal{X}$AGra.

This XML dialect allows the declaration of all grammar symbols (terminals and non-terminals) and all (inherited and synthesized) attributes, and also the definition of all semantic rules (to evaluate the attributes, define contextual conditions, and translate the language), which are the elements needed to specify a complete attribute grammar. In addition, it is possible to identify modules or libraries to be imported, and also to define auxiliary functions. It is worth notice that the XML-Schema (XSD) underlying $\mathcal{X}$AGra was developed rigourously taking into account the formal definition of AG and the extra information that a CG needs to produce a complete language processor — that working approach led to a fast design and implementation.

We also envisage a broader field of applications for $\mathcal{X}$AGra representation than just the role of a general interface between editor-like tools and compiler generators. To corroborate that idea, we develop in the context of a language engineering course a powerful tool (called $\mathcal{X}$AGraAl) for AG analysis and transformation; $\mathcal{X}$AGraAl takes as input a $\mathcal{X}$AGra AG.

Due to its role in the conception of $\mathcal{X}$AGra, attribute grammars are formally defined in section 2. Then, section 3 is devoted to the introduction of $\mathcal{X}$AGra dialect (highlighting its derivation from the AG definition). Section 4 introduces $\mathcal{X}$AGraAl tool and summarizes its functionality. Conclusion and future work sum up the paper in section 5.

2 Attribute Grammars

A *Context-Free Grammar* (CFG) is formally defined by the following tuple:

$$G = (T, N, S, P)$$

where:
T is the set of terminal symbols that define the alphabet for the language;
N is the set of nonterminal symbols;
$S \in N$ is the start symbol of the grammar and
P is a set of productions.

A production, also known as derivation rule, is composed of a *Left-Hand Side* (LHS), with LHS $\in N$; and a *Right-Hand Side* (RHS), with RHS $\subseteq (N \cup T)^*$.

An Attribute Grammar is based on a CFG. It associates: a set, $A(X)$, of attributes with each symbol X in the vocabulary ($V = T \cup N$) of G; a set, $R(p)$, of evaluation rules with each production $p \in P$; and a set, $C(p)$, of contextual conditions with each production $p \in P$.
So an attribute grammar is formally defined as the following tuple:

$$AG = (G, A, R, C, \mathcal{T})$$

where
$A = \bigcup_{X \in (N \cup T)} A(X)$ is the set of all the attributes;
$R = \bigcup_{p \in P} R(p)$ is the set of evaluation rules for all the productions;
$C = \bigcup_{p \in P} C(p)$ is the set of contextual conditions for all the productions and
$\mathcal{T} = \bigcup_{p \in P} \mathcal{T}(p)$ is the set of translation rules for all the productions.
Each attribute has a type, and represents a specific property of symbol X; we write $X.a$ to indicate that attribute a is an element of $A(X)$. For each $X \in (N \cup T)$, the set of attributes of X is splitted into two disjoint sets: $A(X) = Inh(X) \cup Syn(X)$, respectively the *inherited* and the *synthesized* attributes.
Each $R(p)$ is a set of formulas

$$X.a = func(..., Y.b, ...)$$

that define how to compute, in the precise context of production p, the value of each attribute a as a function of the value of other attributes b, where each defined attribute a should be a synthesized attribute associated with the nonterminal in the lefthand side or an inherited attribute associated with a nonterminal in the righthand side:

$$a \in (Syn(X_0) \cup Inh(X_i)), i \geq 1$$

and each used attribute b should be an inherited attribute associated with the nonterminal in the lefthand side or a synthesized attribute associated with a symbol in the righthand side:

$$b \in (Inh(X_0) \cup Syn(X_i)), i \geq 1$$

Each $C(p)$ is a set of predicates

$$pred(..., X.a, ...)$$

describing the requirements that must be satisfied in the precise context of production p. Each predicate, checked for the actual value of the argument attributes (any synthesized or inherited attribute that occurs in that context can be an argument), must hold a `true` value, so that the production is meaningful (is valid from a semantic point of view).
Each $\mathcal{T}(p)$ is a set of procedures

$$proc(..., X.a, X.b, ...)$$

that use the value of attributes available in the context of production p (preferably the synthesized attributes of production, but not restricted to) to produce, or generate, the output of the language processor.

Many times, and many authors, do not consider $C(p)$ and $\mathcal{T}(p)$ separate from $R(p)$: they consider an AG as a triplet

$$AG = (G, A, R)$$

In these cases, contextual conditions and translation rules are defined as boolean functions that associate a truth value with special boolean attributes and produce the desired action (contextual constraint verification or output building) as a side effect.

In summary, AGs are a formal and practical way to develop any kind of programming language. The possibility to use attributes to store and spread information through the processing phase, makes easier the derivation of all modules needed to compile a language [4], and hence, it is faster to get the desired output.

3 $\mathcal{X}$AGra language

In this section is defined an XML dialect to cope with attribute grammars. We gave it the name of $\mathcal{X}$AGra, which stands for ***X**ML dialect for **A**ttribute **Gra**mmars*. From here on, this XML notation will be referred to as $\mathcal{X}$AGra.
$\mathcal{X}$AGra denotes the abstract representation of an AG. Its notation, here defined, is mainly based on the definition of AG presented in Section 2, but it also borrows parts from the notations inherent to various AG-based compiler generator tools.
One of the standardized ways to define a new XML dialect is the creation of a schema, using the standard XML Schema Definition (XSD) language. For the sake of space, the integral textual definition of $\mathcal{X}$AGra's schema is not presented, and for reasons of visibility and readability, the complete drawing of the schema is broken into several important sub-parts. These sub-parts are explained in the present section. Figures 1 to 7 are used to support the explanation of the dialect.

$\mathcal{X}$AGra's root element was defined as `attributeGrammar`. This element has a single attribute, `name`, whose objective is to store the name of the grammar, or the language that the grammar defines; and is a sequence of several elements. These elements represent components of the formal definition of an AG, incremented with extra parts related to the usage of AG-based compiler generators.
Table 1 defines a relation of inclusion between the $\mathcal{X}$AGra notation elements and the components that constitute the formal definition of an AG, which is recovered next:

$$AG = (T, N, S, P, A, R, C, \mathcal{T})$$

The relations depicted in Table 1 give an overview about the information that each element of $\mathcal{X}$AGra notation will store. The following sections will describe with more detail such elements and the information they store.
Listing 1.1 presents a fragment of a grammar that computes the age of a set of students. This example is used to compare the concrete notation of a compiler generator to the XML fragments that are shown in the sequent figures.

Table 1. Derivation of $\mathcal{X}$AGra Notation From the Formal Definition of AG

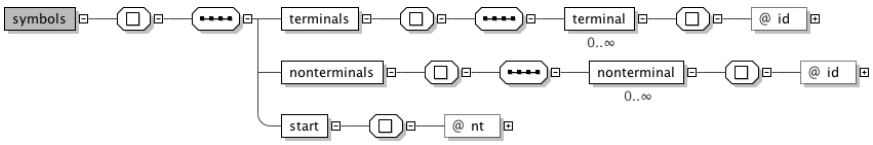
$\mathcal{X}$AGra Element $\supseteq$ AG Components	
symbols	T, N, S
attributesDecl	A
semanticProds	$P, R, C, \mathcal{T}$
importations	$\emptyset$
functions	$\emptyset$

Listing 1.1. Example of Students Grammar

```
language StudentsGra {
   lexicon{
      Name     [A-Z][a-z]+
   ...
   }
   attributes
      int  STUDENTS.sum;
   ...
   rule Students_1 {
      STUDENTS ::=  STUDENT STUDENTS compute {
         STUDENTS.sum = STUDENTS[1].sum + STUDENT.age;
      };
   }
   ...
   method user_Definitions {
      import java.util.ArrayList
      public int sum(int x, int y){
        return x+y;
      }
   }
}
```

3.1 Element *symbols*

Figure 1 presents the schema for the element symbols. As the name suggests, this element contains the declaration of the grammar's vocabulary.
It is composed of a sequence of three elements: terminals, nonterminals and start.
The element terminals is a sequence of zero or more elements named terminal, which, in its turn, has one attribute, id, used to store the name of a terminal symbol. This attribute is an identifier, hence any instance of it, must be different from the others, and must be always instantiated. Besides the information kept on the attribute, this element has a textual content where the respective *Regular Expression* (RE) can be declared.
The element nonterminals has similar structure. The difference lays on the fact that it represents a sequence of zero or more elements nonterminal which have no textual content. The attribute id has the same purpose as the attribute with the same name in the element terminal.
Finally, the element start has a single attribute named nt. This attribute is used to refer the nonterminal (already defined in the $\mathcal{X}$AGra specification), correspondent to the start symbol (or Axiom) of the AG.

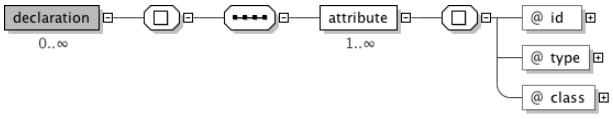



```
<symbols>
    <terminals>
        <terminal id="name">[A–Z][a–z]+</terminal>
    </terminals>
    <nonterminals>
        <nonterminal id="students" />
    </nonterminals>
    <start nt="students" />
</symbols>
```

Fig. 1. $\mathcal{X}$`AGra` Schema – Element **Symbols**: definition and example

3.2 Element *attributesDecl*

This element is composed of a sequence of zero or more elements `declaration`. For the sake of readability, Figure 2 only depicts the structure of the element `declaration`, which is a sequence of one or more elements `attribute`. This one has three mandatory attributes: $i)$ `id` – stores the name of the attribute being declared. Any kind of text can be used to define it, but it is always better to use the following notation: $X.a$, where X is the name of a symbol in $T \cup N$ and a is the name of an attribute in $A(X)$. As it is an identifier, it must be different from all other identifiers on the specification; $ii)$ `type` – stores the data type of the current attribute value and $iii)$ `class` – defines the class of the attribute. It must be one of: InhAttribute, SyntAttribute and IntrinsicValueAttribute.




```
<attributesDecl>
    <declaration>
        <attribute id="students.sum" type="int" class="SyntAttribute" />
    </declaration>
</attributesDecl>
```

Fig. 2. $\mathcal{X}$`AGra` Schema – Element **Attribute Declarations**: definition and example

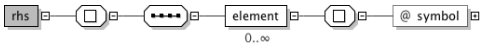
3.3 Element *semanticProds*

The element `semanticProds` represents the structure to define productions and associated semantic rules in $\mathcal{X}$`AGra` specifications. This structure is composed of a sequence of zero or more elements `semanticProd`. Each `semanticProd` has one single attribute, `name`, used to store the mandatory name of the production, as an identifier.
Element `semanticProd` has three direct descendants: `lhs`, `rhs`, `computation`, whose structure is explained in the next paragraphs and that are depicted in Figures 3, 4 and 5.
Element `lhs` (Figure 3) is used to refer to the nonterminal symbol on the `LHS` of the production. This element has a single attribute, `nt`, to refer to an existent `nonterminal`.


```
1 <lhs nt="students" />
```

Fig. 3. $\mathcal{X}$`AGra` Schema – Element **Semantic Productions**: `LHS` definition and example

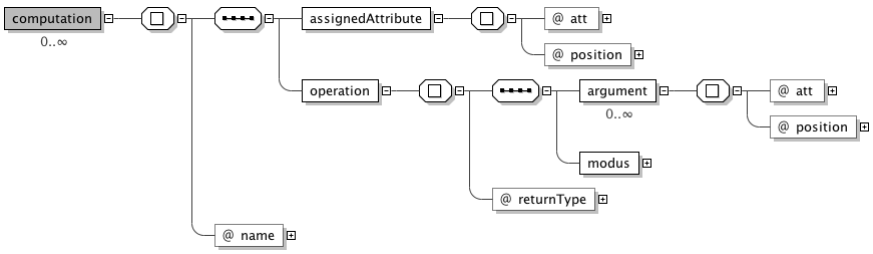
Element `rhs` (Figure 4), stores the nonterminals on the `RHS` of a production. It is composed of a sequence of zero or more elements `element`. For this purpose, each `element`, has a single attribute, `symbol`, which is mandatory and represents a reference to a terminal or nonterminal symbol, already instantiated in the initial `symbols` structure.




```
1 <rhs>
2     <element symbol="student" />
3     <element symbol="students" />
4 </rhs>
```

Fig. 4. $\mathcal{X}$`AGra` Schema – Element **Semantic Productions**: `RHS` definition and example

Element `computation` (Figure 5) is the last child of the element `semanticProds`. It represents an hard concept of `AG`s: the semantic rules.
This element has one attribute, `name`, used to give a name to the computation being declared. This attribute, despite being mandatory, is not a unique identifier: different computations can have equal names.
The structure of `computation` represents a pure abstraction of what is a semantic rule in an `AG` definition: the attribute to which a value is assigned, and the operation that




```
1 <computation name="getTheSum">
2     <assignedAttribute att="students.sum" position="0" />
3     <operation returnType="int">
4         <argument att="student.age" position="1" />
5         <argument att="students.sum" position="2" />
6         <modus> $1 + $2 </modus>
7     </operation>
8 </computation>
```

Fig. 5. $\mathcal{X}$AGra Schema – Element **Semantic Productions**: `Computation` definition and example

computes this value. Thus, the element `computation` has two children: the elements `assignedAttribute` and `operation`.

Element `assignedAttribute` is composed of two mandatory attributes: `att`, which is used to refer to an attribute; and `position`, which is a number that identifies the position of the symbol associated to the attribute in the list of elements of the production. That is, if the attribute is connected to the `LHS`, then the value for `position` must be 0. If the associated symbol belongs to the `RHS`, then its value should correspond to the position that the symbol occupies in the `RHS` sequence of symbols, starting with 1.

The element `operation` aggregates a sequence of zero or more elements `argument` and a single element `modus`. In addition to the elements, it has an attribute, `returnType`, used to store the data type of the value returned by the operation.

Elements `argument` are, in all aspects, equal to the `assignedAttribute` element. Each one has two attributes with the same name and the same semantic value underlaying, therefore they are used to refer to previous declared attributes. The difference is on the fact that this time, the attributes referenced are those used to compute the value in the operation.

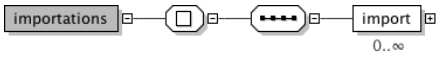
The last element, `modus`[3], which is a simple text field to write the expression used to compute the value. Somehow, in this element's text, a reference to the argument attributes should be made. An example (and the convention established) is using $\$x$, where $x > 0$ is the position of the attribute in the sequence of arguments.

The next two simple parts extend the mathematical definition of `AG`s to the abstract language of any compiler generator based on `AG`s.

[3] *modus* is a latin expression for *way* (of computing something, in our case)

3.4 Element *importations*

Figure 6 presents the structure to declare the importation of packages or programming language modules that can be necessary for the computation of all attributes. The element `importations` is a sequence of zero or more elements `import`. Each of these elements `import` is a simple text container, where the name of the package or module should be written.




```
1 <importations>
2     <import>
3         java.util.ArrayList
4     </import>
5 </importations>
```

Fig. 6. $\mathcal{X}$`AGra` Schema – Element **Importations**: definition and example

3.5 Element *functions*

In a very similar way, element `functions` is a sequence of zero or more `function` elements. Each `function` element has a mandatory attribute, `name`, used to store the name of the function. This element is defined as a text container, in order to be possible the definition of a concrete function. The code of the function must be written in the target programming language like `Java` or other.


```
1 <functions>
2     <function name="sum">
3         public int sum(int x, int y){
4           return x + y;
5         }
6     </function>
7 </functions>
```

Fig. 7. $\mathcal{X}$`AGra` Schema – Element **Functions**: definition and example

$\mathcal{X}$`AGra`'s schema is now completely defined and explained, revealing the universality needed to store any `AG` for any `AG`-based compiler generator.

4 Application Example

In this section we give a brief introduction to $\mathcal{X}$AGraAl, a *Grammar Analyzer and Transformation tool* that computes dependencies among symbols, some grammar metrics, and grammar slices for a given criterion; moreover, $\mathcal{X}$AGraAl can also derive, from the original, new shorter grammars combining slices or removing useless productions (similar to re-factoring a program). $\mathcal{X}$AGraAl takes as input an AG written in $\mathcal{X}$AGra; thus, the presentation of this tool is precisely aiming at illustrating the applicability of $\mathcal{X}$AGra as a universal AG specification language.
$\mathcal{X}$AGraAl is a platform independent tool, developed using Java. To parse the input it is used the Java Architecture for XML Binding (JAXB) [5] and Java API for XML Processing (JAXP) [6]. JAXB simplifies the access to the XML document from a Java program by presenting the document in Java format. All JAXB implementations provide a tool called a *binding compiler* to bind a schema (the way the binding compiler is invoked can be implementation-specific). *Binding a schema*, the first step in this processing approach, means generating a set of Java classes that represents the schema. Those classes are then instantiated during the parsing.
While parsing a $\mathcal{X}$AGra grammar using JAXB, $\mathcal{X}$AGraAl builds the identifiers table (IdTab) where it collects all grammar symbols and attributes; each identifier is associated with all its characteristics extracted or inferred from the source document. The identifiers table — that can be pretty-printed in HTML — complemented by the dependence graph (DG) — also printable using Dot and GraphViz — constitute the core of the tool. Traversing those internal representation structures, it is possible to implement the other $\mathcal{X}$AGraAl functionalities:

- Metrics, to assess grammar quality;
- Slicing, to ease the analysis producing sub-grammars focussed in a specific symbol or attribute;
- Re-factoring, to optimize grammars generating smaller and more efficient versions.

Metrics are organized in three groups of assessment parameters:

- Size metrics, that measure the number of symbols, productions, and so on (grammar and parser sizes);
- Form metrics, that describe the recursion pattern and measure the dependencies between symbols (the grammar complexity);
- Lexicographic metrics, that qualify the clearness/readablity of grammar identifiers, based on a domain ontology.

Slicing operation builds partial grammar with the elements that derive in zero or more steps on the criterion (backward slicing), or that are reachable from the criterion (forward slicing). The criterion can be either a symbol or an attribute. Slices are usually presented as paths over the dependence graphs. Figures 8 (a) and (b) illustrate a forward and a backward slice w.r.t the symbol *age*.
Re-factoring is a not so usual functionality that transforms the original grammar into a minimal one, removing all the *useless productions*. Another transformation also provided is the generation of a new grammar combining forward and backward slices with respect to the same symbol (see Figure 8 (c)).

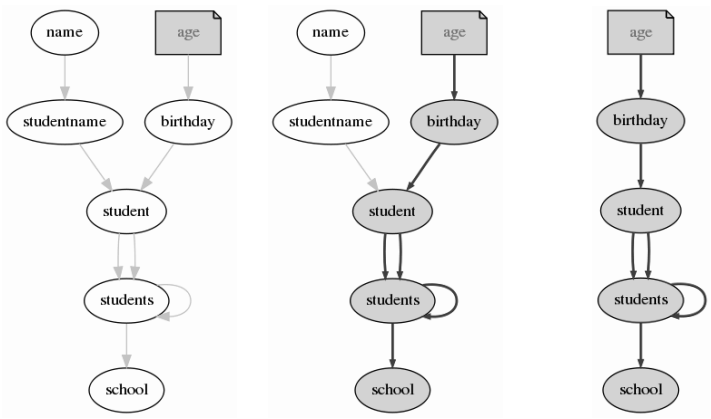


Fig. 8. Slices with respect to symbol *age*: (a) Forward slice; (b) Backward slice and (c) Combination of Forward and Backward slices

5 Conclusion

In the context of our Visual$\mathcal{LISA}$ project, we felt strongly impelled to use a universal meta-language for attribute grammars description in order to translate the graphical specification drawn in Visual$\mathcal{LISA}$, instead of producing a translation specific for a given compiler generator. As we did not find any notation with this tool-independence characteristic, we realized that a new one must be defined. As a mandatory requirement we stated that it should be used by our AG development environment, as well as it should be the *lingua-franca* of grammar-based tools. To satisfy the first part of the requirement it should allow to describe all elements present in a concrete AG that specifies the syntax of a language, and also those complementary definitions that permit the generation of a compiler for that language. To satisfy the second part of the above statement, the new meta-language should be supported on XML.

Conceived to annotate unstructured documents in a way independent of their future processing, XML immediately became the universal interchangeable data representation for assuring systems' interoperability. So our decision to design a specific XML-Schema for AGs, was obvious. Along the paper, we have shown how that design, was conducted systematically by the formal definition of attribute grammar.

Besides the presentation of $\mathcal{X}$AGra, the first goal of this paper, we also introduced $\mathcal{X}$AGraAl, a tool (completely independent of the one that motivates the conception of $\mathcal{X}$AGra) that supports grammar analysis taking as input a $\mathcal{X}$AGra grammar. The objective was to illustrate an applicability of this new XML dialect for AGs description.

Visual$\mathcal{LISA}$ environment is now generating $\mathcal{X}$AGra to translate a visual AG into a textual format usable by a compiler generator, or similar tool. The adaptation of the original $\mathcal{LISA}$ generator to the new one, that should now be called VisualAG, was an easy task performed very fast. However, for each generator that we want to couple to

VisualAG, it will be necessary to develop a $\mathcal{X}$AGra uploader, this is, a translator from $\mathcal{X}$AGra to the specific notation of the tool under consideration. As future work, the following translators are planed: $\mathcal{X}$AGra into LISA (a traditional LR parser generator); $\mathcal{X}$AGra into AntLR (an LL parser generator, based on an extended BNF grammar); $\mathcal{X}$AGra into Eli (an LR parser generator with special constructors). Also a translator from $\mathcal{X}$AGra to Yacc, could be a challenging project.

The creation of these translators is possible and easy. We are sure of this, because $\mathcal{X}$AGraAl showed its feasability. Moreover, $\mathcal{X}$AGraAl's front-end (the parser and semantic analyser that reads the $\mathcal{X}$AGra input and transforms it into an internal representation for further processing) would be similar to the core of these translators, so it can be reused.

To conclude, we claim that: (i) $\mathcal{X}$AGra is abstract and universal and (ii) this dialect was not crafted to be pleasant for human reading. Concerning the first point, we base that statement on the fact that $\mathcal{X}$AGra was derived from the formal and complete definition of Atrribute Grammar. Also, the dialect is abstract because it is completely independent from the concrete syntax of any compiler generator (CG). The second point is obvious: reading XML documents, although possible, is a cumbersome task for humans; the purpose of this dialect (generated by a tool) is to be processed by tools like $\mathcal{X}$AGraAl or the translators we plan for further work.

References

1. Knuth, D.E.: Semantics of context-free languages. Theory of Computing Systems **2**(2) (June 1968) 127–145
2. Oliveira, N., Pereira, M.J.V., da Cruz, D., Henriques, P.R.: VisualLISA. Technical report, Universidade do Minho (February 2009) www.di.uminho.pt/~gepl/VisualLISA/documentation.php.
3. Mernik, M., Lenič, M., Avdičaušević, E., Žumer, V.: LISA: An interactive environment for programming language development. Compiler Construction (2002) 1–4
4. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition). Addison Wesley (August 2006)
5. GlassFish: Java architecture for XML binding. https://jaxb.dev.java.net/ (June 2009)
6. GlassFish: Java API for XML processing. https://jaxp.dev.java.net/ (June 2009)

Extending the Learning Object definition to represent Programming Problems

José Paulo Leal[1] and Ricardo Queirós[2],

[1] CRACS & DCC-FCUP, University of Porto, Portugal
zp@dcc.fc.up.pt
[2] CRACS & DI-ESEIG/IPP, Porto, Portugal
ricardo.queiros@eu.ipp.pt

Abstract. The present generation of eLearning platforms values the interchange of learning objects standards. Nevertheless, for specialized domains these standards are insufficient to fully describe all the assets, especially when they are used as input for other eLearning services. To address this issue we extended an existing learning objects standard to the particular requirements of a specialized domain, namely the automatic evaluation of programming problems. The focus of this paper is the definition of programming problems as learning objects. We introduce a new schema to represent metadata related to automatic evaluation that cannot be conveniently represented using existing standards, such as: the type of automatic evaluation; the requirements of the evaluation engine; or the roles of different assets - tests cases, program solutions, etc. This new schema is being used in an interoperable repository of learning objects, called crimsonHex.

Keywords: eLearning, Learning Objects, Content Packaging, Interoperability.

1 Introduction

The majority of the eLearning platforms available today follow a component-oriented architecture. These systems assemble a collection of generic tools - such as forums or multiple choice quizzes - that are considered to be useful for all learning areas. Despite their success, they have also been target of criticism: their tools are too general and they are difficult to integrate with other eLearning systems [1]. These issues led to recent initiatives to adapt Service Oriented Architecture (SOA) [2] to eLearning. Apart from the systems integration, other problems arise related with the standardization of eLearning content. The existing standards are too generic and not adequate to specific domains, such as the definition of programming problems.

This paper focuses on a definition of programming problems as learning objects (LO) adequate to the interoperability of services in the area of automatic evaluation of programming problems. This new definition represents also a new application profile for learning objects based on Instructional Management Systems (IMS) specifications and extended to accommodate domain specific issues. This definition is being used in

a European research project called EduJudge, which aims to integrate a collection of problems created for programming contests into an effective educational environment. The eLearning system resulting from the EduJudge project includes different types of services hence a precise definition of programming problems as learning objects is essential to ensure interoperability among them.

The remainder of this paper is organized as follows: Section 2 traces the evolution of LO standards with emphasis on schema languages. In the following section we present a new application profile based on IMS specifications to represent programming problems. In this section we also detail the combination of two different types of validation languages: grammar and rule-based schema languages. Then, we present a case study regarding the use of the new application profile in crimsonHex, a repository of specialized learning objects. Finally, we conclude with a summary of the main contributions of this work and a perspective of future research.

2 State of Art

The evolution of eLearning systems in the last two decades was impressive. In their first generation, eLearning systems were developed for a specific learning domain and had a monolithic architecture [1]. Gradually, these systems evolved and became domain-independent, featuring reusable tools that can be effectively used virtually in any eLearning course. The systems that reach this level of maturity usually follow a component-oriented architecture in order to facilitate tool integration. An example of this type of system is the Learning Management System (LMS) that integrate several types of tools for delivering content and for recreating a learning context (e.g. Moodle, Sakai).

The present generation values the interchange of learning objects and learners' information through the adoption of new standards that brought content sharing and interoperability to eLearning. Standards can be viewed as "documented agreements containing technical specifications or other precise criteria to be used consistently as guidelines to ensure that materials and services are fit for their purpose" [3]. In the eLearning context, standards are generally developed with the purpose of ensuring interoperability and reusability in systems. In this context, several organizations [4, 5, 6] have develop specifications and standards in the last years [7]. These specifications define, among many others, standards for eLearning content [8, 9, 10] and interoperability [11, 12].

As said before, current LO standards are quite generic and not adequate to specific domains, such as the definition of programming problems. The most widely used standard for LO is the IMS Content Packaging (IMS CP). This content packaging format uses an XML manifest file wrapped with other resources inside a zip file. The manifest includes the IEEE Learning Object Metadata (LOM) standard to describe the learning resources included in the package. However, LOM was not specifically designed to accommodate the requirements of automatic evaluation of programming problems. For instance, there is no way to assert the role of specific resources, such as test cases or solutions. Fortunately, IMS CP was designed to be straightforward to

extend, meeting the needs of a target user community through the creation of application profiles.

When applied to metadata the term Application Profile generally refers to "the adaptation, constraint, and/or augmentation of a metadata scheme to suit the needs of a particular community" [13]. A well know eLearning application profile is SCORM [14] that extends IMS CP with more sophisticated sequencing and Contents-to-LMS communication.

The creation of application profiles aims to meet the needs of the target user community, aid integration and enhance interoperability between tools and services of the community. The creation is based in one or more of the following approaches:

- Selection of a core sub-set of elements and fields from the source schema;
- Addition of elements and/or fields (normally termed extensions) to the source schema, thus generating the derived schema;
- Substitution of a vocabulary with a new, or extended vocabulary to reflect terms in common usage within the target community;
- Description of the semantics and common usage of the schema as they are to be applied across the community.

Following this extension philosophy, the IMS Global Learning Consortium (GLC) upgraded the Question & Test Interoperability (QTI) specification [10]. QTI describes a data model for questions and test data and, from version 2, extends the LOM with its own metadata vocabulary. QTI was designed for questions with a set of pre-defined answers, such as multiple choice, multiple response, fill-in-the-blanks and short text questions. It supports also long text answers but the specification of their evaluation is outside the scope of the QTI. Although long text answers could be used to write the program's source code, there is no way to specify how it should be compiled and executed, which test data should be used and how it should be graded. For these reasons we consider that QTI is not adequate for automatic evaluation of programming exercises, although it may be supported for sake of compatibility with some LMS. Recently, IMS GLC proposed the IMS Common Cartridge [15] that bundles the previous specifications and its main goal is to organize and distribute digital learning content.

All these standards are described by schema languages, most often using the XML Schema Definition language (XSD). This language overcame DTD limitations and provided several advanced features, such as, the ability to build new types derived from basic ones, manage relationships between elements (similar to relational databases) and combine elements from several schemata.

In spite of its expressiveness, XSD lacks features to describe constraints on the XML document structure. For instance, there is no way to specify dependencies between attributes, or to select the content model based on the value of another element or attribute. To address these issues several schema languages were proposed, such as, RELAX NG [16] (based on TREX [17] and RELAX [18]), DSD (Document Structure Description) [19] and Schematron [20]. The Schematron language provides a standard mechanism for making assertions about the validity of an XML document using XPath expressions and can be easily combined with W3C XML Schema documents.

3 Application profile

Based on the previous approaches to create a new eLearning application profile, we defined programming problems as learning objects by extending the IMS CP specification. An IMS CP learning object assembles resources and metadata into a distribution medium, in our case a file archive in zip format, with its content described in a file named `imsmanifest.xml` in the root level. The manifest contains four sections: metadata, organizations, resources and sub-manifests. The main sections are metadata, which includes a description of the package, and resources, containing a list of references to other files in the archive (resources), as well as dependencies among them.

Metadata information in the manifest file usually follows the IEEE LOM schema, although other schemata can be used. These metadata elements can be inserted in any section of the IMS CP manifest. In our case, the metadata that cannot be conveniently represented using LOM is encoded in elements of a new schema – EduJudge Meta-Data (EJ MD) - and included only in the metadata section of the IMS CP. This section is the proper place to describe relationships among resources, as those needed for automatic evaluation and lacking in the IEEE LOM. The compound schema can be viewed as a new application profile that combines metadata elements selected from several schemata. The structure of the archive, acting as distribution medium and containing the programming problem as a LO, is depicted in Fig. 1.

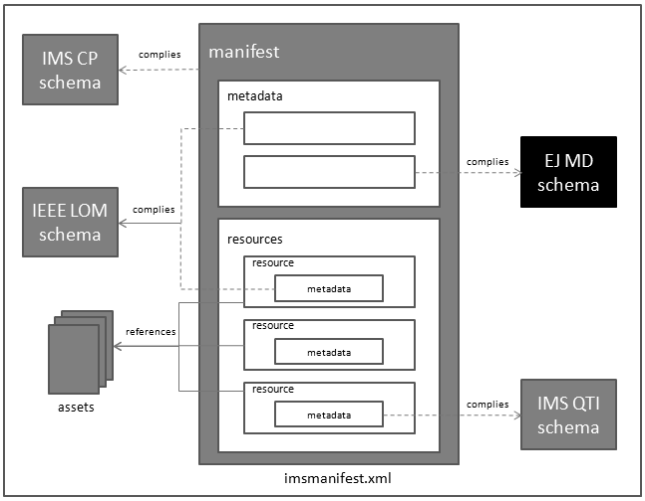


Fig. 1. Structure of a programming problem as a learning object.

The archive contains several files represented in the diagram as grey rectangles. The manifest is an XML file and its elements' structure is represented by white rectangles. Different elements of the manifest comply with different schemata packaged in the same archive, as represented by the dashed arrows: the manifest root element complies with the IMS CP schema; elements in the metadata section may comply either with IEEE LOM or with EJ MD; metadata elements within resources

may comply either with IEEE LOM or IMS QTI. Resource elements in the manifest file reference assets packaged in the archive are represented in solid arrows.

The IMS CP specification is defined by a W3C XML Schema Definition (XSD). The schema describes which elements may exist in the document manifest and how those elements may be structured. Instance documents can be validated using this XSD schema. In our application profile we used elements from several schemata and namespaces were used to avoid name clashes. In the EJ MD specification, the namespaces, filenames and namespace prefixes of XML instances are as follows:

Table 1. Schemata in the new Application Profile.

Specification	Namespace	Filename
IMS CP	http://www.imsglobal.org/xsd/imscp_v1p1	imscp_v1p1.xsd
IEEE LOM	http://www.imsglobal.org/xsd/imsmd_v1p2	imsmd_v1p2.xsd
IMS QTI	http://www.imsglobal.org/xsd/imsqti_v1p1	imsqti_v1p1.xsd
EJ MD	http://www.edujudge.eu/ejmd_v2	ejmd_v2.xsd

These references will be used for on-line validation, to conform to IMS CP Best Practice Document – to prefer online references on the IMS website, rather than static XSD files in the LO package, as they will be the most up-to-date specifications.

3.1 The EduJudge schema

The corner stone of this definition of programming problems as learning objects is automatic evaluation. Consequently, this definition assumes the existence of a component responsible for evaluating learners' attempts based on the learning object and producing a result. Moreover, it needs also to assume an evaluation model supported by the evaluator. After considering several possible alternatives we decided on a single and simple evaluation model following three steps:

1. the evaluator receives:
    a. a reference to the learning object with a programming problem;
    b. an attempt to solve it - a single file, a program or an archive containing files of different types (e.g. JAR, WAR);
    c. a reference to the learner submitting the attempt.
2. the evaluator processes this data as follows:
    a. loads the learning object from a repository using its reference;
    b. uses the assets available in the LO (static tests, generated tests, unit tests, etc.) according to their role;
    c. produces a result (correction , classification and feedback) that may depend on the learner's reference;
    d. stores the result for future incremental feedback to the same learner.
3. the evaluator returns the result immediately or with a short delay.

Assuming this simple model, the learning object metadata simply assigns a role to each asset. It is the responsibility of the evaluation component to use each asset appropriately according to its role.

To represent programming problems as learning objects, able to be evaluated according to model we just described, we extended the metadata of the IMS CP, as foreseen in this specification. New metadata can be inserted in several points of the manifest. Based on the available choices we decided to place different types of metadata in the following extension points:

- **Domain metadata** (EJ MD), related to the automatic evaluation, in IMS CP `manifest/metadata` element;
- **Resource metadata** (IEEE LOM), independent from their use in automatic evaluation, within the IMS CP `manifest/resource/file/metadata` elements (without any domain metadata) and linked by the domain data through IDREF attributes.

The domain metadata shown in Figure 2 is divided in three categories: the general category describes generic metadata and recommendations; the presentation category describes metadata on resources that are presented to the learner (e.g. description and skeleton resources); the evaluation category describes the metadata on resources used to evaluate the learner's attempts and provide feedback.

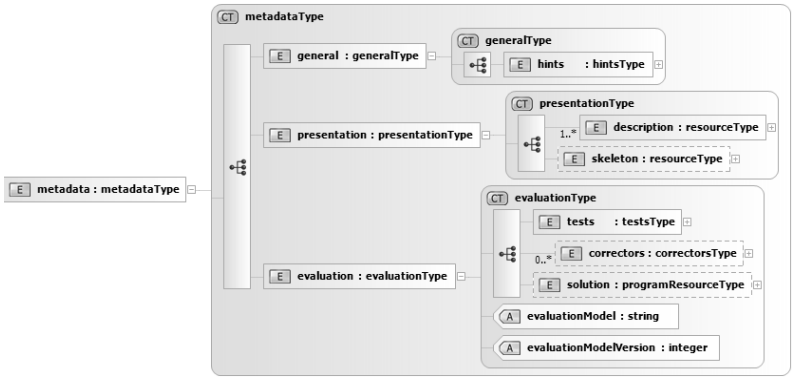


Fig. 2. The domain metadata of the EJ MD specification

The IMS CP resources section is a collection of resource elements, each one grouping several files. In order to link the EJ MD domain metadata described above, with the related resources, we used the IDREF XML Schema type in the domain metadata to reference the `resource` elements, more precisely, the IMS CP `identifier` attribute, as represented in Fig. 3.

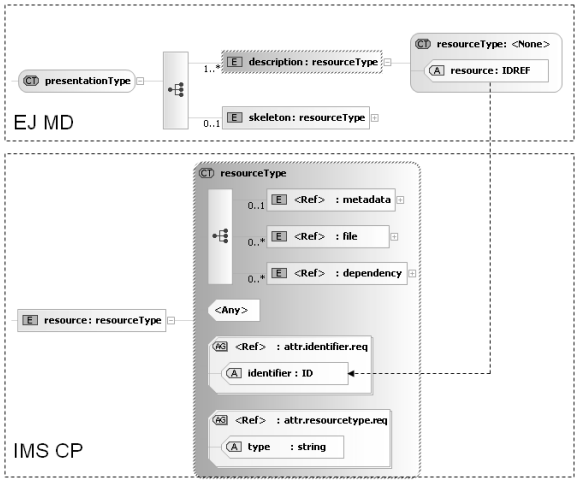


Fig. 3. The binding model of EJ MD domain metadata and the IMS CP resources

3.2 Schematron processing

Despite the expressiveness of XML Schema, there are several situations where it is not possible to validate a document with only this language. For example, the following cases cannot be validating using an XML Schema:

- general type is included only in the `manifest/metadata` element (because of the multiple extension points provided by the IMS CP);
- IDREF type points to a file resource with the appropriate type;
- value of the `imsmd:minimumversion` element is less than the value of the `imsmd:maximumversion` element.

To check this type of constraints we cannot use XML Schema. There are, at least, three options: combine with others schema languages; write code in a programming language to express the additional constraints; use an XSLT/XPath stylesheet. We will use the former, because we want to maintain the solutions based in XML technologies and, if possible, in a single schema document. There are several alternative schema languages, such as RELAX, TREX and Schematron. In this case, we need to use a rule-based validation language in order to find certain patterns in the XML document. A good candidate to this "second level of validation" is Schematron. The last constraint enumerated could be validated with this rule as a separate file:

```
<schema xmlns="http://www.ascc.net/xml/schematron" >
  <pattern name="version validation">
    <rule context="//imsmd:requirement">
      <assert test="imsmd:minimumversion &gt; imsmd:maximumversion">
        ERROR
      </assert>
    </rule>
  </pattern>
</schema>
```

Schematron validation can be used in conjunction with a XML schema validation using two approaches:

- as separate files (using pipeline validation languages [21,22]);
- as a unique file (embedding Schematron rules in the XML Schema).

To simplify the file version management we decided for the second option and used Schematron rules embedded within the `appinfo` elements in the XSD document. However, a W3C XML Schema processor does not validate constraints expressed by the embedded Schematron rules. They need to be extracted from the source schema and concatenated into a new Schematron document. To address this issue we created a stylesheet (`Schematron-Generator.xsl`) to extract embedded Schematron rules from a W3C XML Schema document and merge them into a complete schema. This approach was used in Robertsson work [23] and can be summarized by Fig. 4.

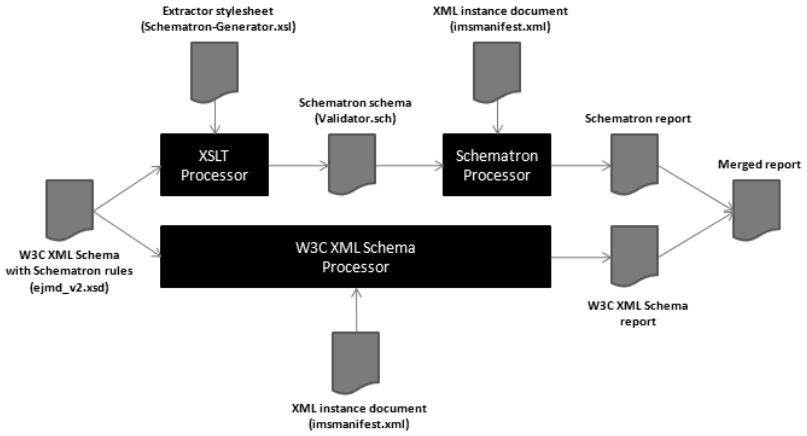


Fig. 4. Validation of XML files by a W3C XML Schema with Schematron rules.

Since Schematron rules are built using XPath and XSLT functions, the Schematron processor depicted in the previous figure is based on a XSLT processor.

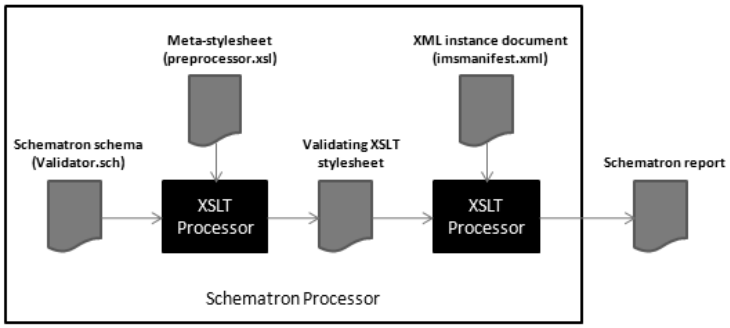


Fig. 5. Schematron processing.

To perform this validation we used an implementation of a Java API for the Schematron language [24] that organizes the Schematron processing in two steps, as shown in Fig. 5:

- The Schematron schema is transformed into a validating XSLT stylesheet by a meta-stylesheet provided by the API.
- The validating stylesheet is then used on the XML instance document and the result will be a report based on rules/assertions of the Schematron schema.

With this approach we can benefit from the combination of these two powerful validation languages and many of the constraints that previously had to be checked in the application code can now be abstracted to the schema. However it should be noticed that in time critical applications the overhead of processing the embedded Schematron rules may be unaffordable.

4 Case Study

In this section we describe the integration of the proposed programming problem definition in a specialized and interoperable repository of LO named crimsonHex. The crimsonHex repository is being used in a European research project called EduJudge that aims to open Valladolid on-line judge (http://uva.onlinejudge.org/) to secondary and higher education, benefiting from its considerable collection of programming problems from international and worldwide ACM-ICPC competitions.

The integration of the EduJudge schema in crimsonHex and the feedback from other partners in the project were crucial to evaluate the usefulness of the proposed IMS application profile. In the remainder of this section we make a succinct description of the repository with emphasis on XML storage and validation. Details on the implementation of crimsonHex can be found elsewhere [25].

4.1 Repository components

In the design of crimsonHex we set some initial requirements, in particular, to be simple and efficient. Simplicity is the best way to promote the reliability and efficiency of the repository. In fact, the core operations of the repository are uploading and downloading LO - ZIP archives - which are inherently simple operations that can be implemented almost directly over the transport protocol. Other features may need a more elaborate implementation but do not require the same reliability and efficiency of the core features. The architecture of crimsonHex repository is divided in three main components:

- **the Core** exposes the main features of the repository, both to external services, such as the LMS and the Evaluator Engine, and to internal components - the Web Manager and the Importer;
- **the Web Manager** allows the creation, revision, , uploading/downloading of LOs and related metadata, enforcing compliance with controlled vocabularies;
- **the Importer** populates the repository with existing legacy repositories.

4.2 XML Storage

Searching LOs in the repository is based on queries on their XML manifests. Since manifests are XML documents with complex schemata we paid particular attention to databases systems with XML support: XML enabled relational databases and Native XML Databases (NXD).

XML enabled relational databases are traditional databases with XML import/export features. They do not store internally data in XML format hence they do not support querying using XQuery. Since queries in this standard are a DRI recommendation this type of storage is not a valid option. In contrast, NXD uses the XML document as fundamental unit of (logical) storage, making it more suitable for data schemata difficult to fit in the relational model. Moreover, using XML documents as storage units enables the following standards:

- XPath for simple queries on document or collections of documents;
- XQuery for queries requiring transformational scaffolding;
- SOAP, REST, WebDAV, XmlRpc and Atom for application interface;
- XML:DB API (or XAPI) as a standard interface to access XML datastores.
- XSLT to transform documents or query-results retrieved from the database.

We analysed several open source NXD, including SEDNA, OZONE, XIndice and eXist, Only eXist implements the complete list of the features enumerated above, which led us to select it as the storage component of crimsonHex. It has also two important features [26] worth mentioning: support for collections, to structure the database in groups of related documents and automatic indexes to speed up the database access**.**

4.3 Validation levels

The crimsonHex is a repository of specialized learning objects. To support this multi typed content the repository must have a flexible LO validation feature. The eXist NXD supports implicit validation on insertion of XML documents in the database but this feature could not be used for several reasons: LO are not XML documents (are ZIP files containing an XML manifest); manifest validation may involve many XSD files that are not efficiently handled by eXist; and manifest validation may combine XSD and Schematron validation and this last is not fully supported by eXist.

All LOs stored in crimsonHex must comply with the IMS Package Conformance that specifies it structure and content. This standard also requires the XSD validation of their manifests. For particular domains it is possible to configure specialized validations in crimsonHex by supplying a Java class implementing a specific interface. These validations extend those of the IMS Package Conformance and may introduce new schemata, even using different type definition languages, such as Schematron.

Validations are configured per collection of documents. Thus, different types of specialized LO may coexist in a single instance of crimsonHex. As mentioned before, IMS CP main schema imports many other schemata (more than 30) that according to the IMS Package Conformance must be downloaded from the Internet. This requirement has a huge impact on the performance of the submit function. To

accelerate this function we implemented a cache. A newly stored schema has a time to live of 1 hour. Outdated schemata are reloaded from their original Internet location using a conditional HTTP request that downloads it only if it has effectively changed.

5 Conclusions

In this paper we described the definition of programming problems as learning objects. The main contribution of this work is the extension of an IMS standard to the particular requirements of a specialized domain - the automatic evaluation of programming problems. Although we focused on the automatic evaluation of programming problems, we think that the described approach can be adapted to other learning domains. This new application profile is being used in crimsonHex, an interoperable repository of learning objects.

In its current status the EduJudge Metadata is available for test and download at the following URL `http://www.dcc.fc.up.pt/schemaDoc`. Our future work will be to adapt the schema to support new evaluation models, for instance, programming problems where the evaluator aggregates programs submitted by two or more learners.

Acknowledgments. This work is part of the project entitled "Integrating Online Judge into effective e-learning", with project number 135221-LLP-1-2007-1-ES-KA3-KA3MP. This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

References

1. Dagger, D., O'Connor, A., Lawless, S., Walsh, E., Wade, V.: Service Oriented eLearning Platforms: From Monolithic Systems to Flexible Services (2007)
2. Krafzig, D., Banke, K., Slama, D. Enterprise SOA: Service-Oriented Architecture Best Practices. 1.ed. Estados Unidos da América: Prentice Hall, 2004. ISBN 0131465759
3. Bryden, A.: Open and Global Standards for Achieving an Inclusive Information Society.
4. IMS Global Learning Consortium. URL: http://www.imsglobal.org
5. IEEE Learning Technology Standards Committee. URL: http://ieeeltsc.org
6. ISO/IEC- International Organization for Standardization. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html
7. Friesen, N.: Interoperability and Learning Objects: An Overview of E-Learning Standardization". Interdisciplinary Journal of Knowledge and Learning Objects. 2005.
8. IMS-CP – IMS Content Packaging, Information Model, Best Practice and Implementation Guide, Version 1.1.3 Final Specification IMS Global Learning Consortium Inc., URL: http://www.imsglobal.org/content/packaging.

9. IMS-Metadata - IMS MetaData. Information Model, Best Practice and Implementation Guide, Version 1.2.1 Final Specification IMS Global Learning Consortium Inc., URL: http://www.imsglobal.org/metadata.
10. IMS-QTI - IMS Question and Test Interoperability. Information Model, Best Practice and Implementation Guide, Version 1.2.1 Final Specification IMS Global Learning Consortium Inc., URL: http://www.imsglobal.org/question/index.html.
11. IMS DRI - IMS Digital Repositories Interoperability - Core Functions Information Model, URL: http://www.imsglobal.org/digitalrepositories/driv1p0/imsdri_infov1p0.html.
12. Simon, B., Massart, D., van Assche, F., Ternier, S., Duval, E., Brantner, S., Olmedilla, D., & Miklos, Z. (2005). A Simple Query Interface for Interoperable Learning Repositories. In Proceedings of the WWW 2005 Conference, retrieved March 16, 2006 from http://nm.wu-wien.ac.at/e-learning/interoperability/www2005-workshop-sqi-2005-04-14.pdf.
13. IMS Application Profile Guidelines Overview, Part 1 - Management Overview, Version 1.0. URL: http://www.imsglobal.org/ap/apv1p0/imsap_oviewv1p0.html.
14. ADL SCORM Overview. URL: http://www.adlnet.gov/Technologies/scorm
15. IMS Common Cartridge Profile, Version 1.0 Final Specification. URL: http://www.imsglobal.org/cc/ccv1p0/imscc_profilev1p0.html
16. Clark, J, Murata, M.: RELAX NG Specification, OASIS Committee Specification, December 2001, http://relaxng.org/spec-20011203.html
17. Clark, J.: TREX - Tree Regular Expressions for XML. Thai Open Source Software Center, 2001, http://www.thaiopensource.com/trex/.
18. Murata, M.: RELAX (Regular Language description for XML). INSTAC (Information Technology Research and Standardization Center), 2001, http://www.xml.gr.jp/relax/.
19. Moller, A.: Document Structure Description 2.0, BRICS, 2002, http://www.brics.dk/DSD/dsd2.html.
20. The Schematron, An XML Structure Validation Language using Patterns in Trees, http://www.ascc.net/xml/resource/schematron/schematron.html.
21. Document Schema Definition Languages (DSDL), Working Draft – ISO/IEC 2004. URL: http://www.dsdl.org/.
22. Jelliffe, R.: Schemachine: A framework for modular validation of XML documents, 2002.
23. Robertsson, E.: Combining Schematron with other XML Schema languages, 2002.
24. Hovhannisian, A., Becker, O.: JAVA API for Schematron. URL: http://www2.informatik.hu-berlin.de/~obecker/SchematronAPI/, 2001.
25. Leal, J.P., Queirós, R.: CrimsonHex: a Service Oriented Repository of Specialised Learning Objects. In: ICEIS 2009: 11th International Conference on Enterprise Information Systems, Milan (2009)
26. Meier, W.: eXist: An Open Source Native XML Database. In: NODe 2002 Web and Database-Related Workshops, (2002)

Relational Databases Digital Preservation

Ricardo André Pereira Freitas[1] and José Carlos Ramalho[2]

[1] Faculty of Science – University of Porto
Porto – Portugal
[2] Department of Informatics - University of Minho
Braga – Portugal
rfreitas@fc.up.pt, jcr@di.uminho.pt

Abstract. Digital preservation is emerging as an area of work and research that tries to provide answers that will ensure a continued and long-term access to information stored digitally. IT Platforms are constantly changing and evolving and nothing can guarantee the continuity of access to digital artifacts in their absence.
This paper focuses on a specific family of digital objects: Relational Databases; they are the most frequent type of databases used by organizations worldwide. A neutral format that is hardware and software independent is the key to achieve a standard format to use in digital preservation of relational databases. XML for its neutrality was chosen for this representation of the database.
The presented solution offers a possibility to achieve relational databases preservation. The prototype follows the "Reference Model for an Open Archival Information System" (OAIS).

Key words: Digital Preservation, OAIS, Relational Databases, XML, Digital Objects, Significant Properties

1 Introduction

Nowadays, due to the constant evolution in the hardware and software industry more and more of the intellectual and business information are stored in computer platforms. The main issue lies exactly within these platforms. If in the past there was no need of mediators to understand the analogical artifacts today, in order to understand digital objects, we depend on those mediators (computer platforms). Nothing can guarantee the continuity of access to digital artifacts in their absence [12]. A new problem in the digital universe arises: Digital Preservation.

Accessing the information does not mean a simple access to the bits that all digital objects are made of, rather it means an access to the information understanding what's there. Although digital information can be exactly preserved in its original form by only copying (preserving) the bits, the issue appears when we notice the very fast evolution of those platforms (hardware and software) where the bits can be transformed into something human intelligible [9]. Digital archives are complex structures that without the software and hardware –

which they depend on – the human being, or others, will certainly be unable to experience or understand them [8].

Our work addresses this issue of Digital Preservation and focuses on a specific class of digital objects: Relational Databases [9]. Relational databases are a very important piece in the global context of digital information and therefore it is fundamental not to compromise its longevity (life cycle) and also its integrity, liability and authenticity [15]. These kind of archives are especially important to organizations because they can justify their activities and give us a glimpse about the organization itself. What kind of organization does not have its information system based on IT platforms? So the question is, how can we ensure access for a long-term, to the information of a relational database and understand what's there? The information must be interpretable for those who demand.

2 State-of-the-art

There are known approaches to the problematic of digital preservation – technology preservation, emulation, migration, normalization, encapsulation [12] [18] [20], and more. We intend to research and study the problematic within digital preservation but focusing on a specific family of digital objects: Relational Databases. Before going further lets characterize these digital artifacts.

A database can be defined as a set of information that is structured. In computing, a database is supported by particular program or software, usually called the Database Management System (DBMS), which handles the storage and management of the information. In its essence a database involves the existence of a set of records of data. Normally these records give support to the organization information system; either at an operational (transactions) level or at other levels. For example, obtaining knowledge to help in decision support (Data Warehousing Systems).

The structure of relations and relationships between entities within a database depends on the type of the used model. Our study focuses on the relational model, widely available and certainly the most used. However there are other logical models for databases: the flat model, the hierarchical model, object-oriented model, among others [21].

In digital preservation it is fundamental to establish the significant properties that should be preserved for each class of digital objects. We will try to achieve some consensus over these issues and then analyze some of the current projects on this field of research.

2.1 The Significant Properties

In general the significant properties in a digital object are those that are identified by its community of interest.

Information that indicates the original operating system and the DBMS that used to support the database is important to characterize the environment of the original database. The date of creation of the database and identification of

its creator should also be preserved. This information is identified as technical metadata.

The information in a relational database has a particular structure based in relations usually called tables [5]. Lee Buck [2] and Ronald Bourret [1] on their approaches concerning XML and Databases do not mention any information about the database structure. However, the structure may provide a way of interpreting the data in order to work and extract valid information – knowledge. On one hand we have the data stored in the database and on the other hand its structure. The data contained in the records of the database obviously has to be preserved but through this analysis we conclude that it will be necessary to also preserve the structure of the database [15]. Some structure features considered important for preservation are:

- **TABLES** (Name)
- **COLUMNS** (Name, Type, Size, ...)
- **KEYS** (Primary keys, Foreign keys, ...)

By preserving these elements we are able to preserve all the database structure – relations (tables) and the relationships between them.

There are other features in a database, such as triggers, stored procedures, forms, or other application issues that we should consider whether or not to preserve. These elements differ from the previous ones since they represent the database semantics. Depending on what is considered significant to preserve we may choose to preserve these features or not. If we choose to preserve application issues, such as a form that interacts with the database, it may be enough to preserve its code or it may be necessary to preserve an image of its appearance.

2.2 Current Research

Considering the nature of the digital artifacts that we are addressing – relational databases – there is an European strategy encompassed in the "Planets Project" [13] to enable their long term access. The project adopted the SIARD [17] solution, which is based on the migration of database into a normalized format (XML – eXtensible Markup Language [22]). The SIARD was initially developed by the Swiss Federal Archives (SFA).

Another approach, also based on XML, relies on the main concept of "extensibility" – XML allows the creation of other languages [16] (it can be called as a meta language). The DBML [10] (Database Markup Language) was created in order to enable representation of both **DATA** and **STRUCTURE** of the database. The following diagram (Fig. 1) reflects the schema for this language.

Both approaches (SIARD and DBML) adopt the strategy of Migration of the database to XML, why? A neutral format that is hardware and software (platform) independent is the key to achieve a standard format to use in digital preservation of relational databases. This neutral format should meet all the requirements established by the designated community of interest.

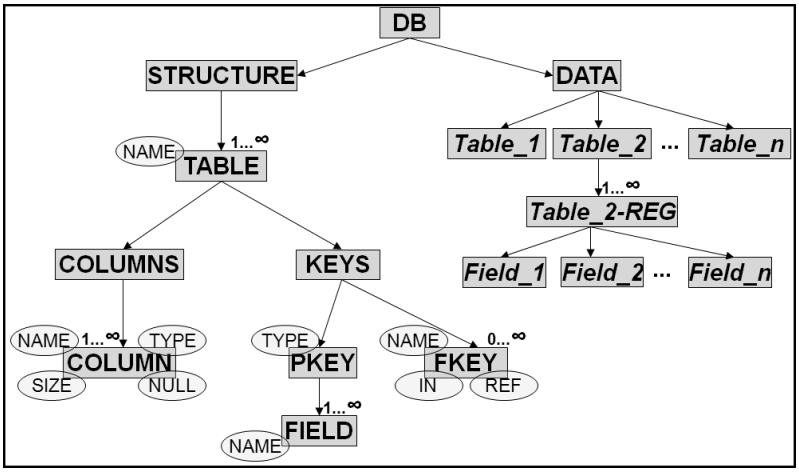


Fig. 1. DBML Schema

2.3 Open Problems

The usage of a normalized format is accepted as being the answer to the problematic of preserving relational databases [10] [17]. By doing this, it is possible to separate the data from its specific database management environment. However, there are still some issues not answered: what significant properties should be preserved? Should the database semantics be preserved? If so, how? How can we ensure authenticity? And how to ensure preservation during the lifecycle of the database (while it evolves)? These are just a few questions that emerge upon this approach.

In order to search for answers on these issues and to pursue other questions or solutions that may emerge, a case study was developed.

3 Possible Solution

Concerning the preservation of relational databases, we adopted an approach that combines two strategies and uses a third technique: migration and normalization with refreshment [9]. The main strategy in our approach is Migration which is carried in order to transform the original database into the new format – DBML [10]. The normalization is crucial to reduce the preservation spectrum to only one format. A third technique (refreshment) will also be needed. The refreshment consists on ensuring that the archive is using media appropriate to the hardware in use throughout preservation [9].

In this case study we used a database on which is not expected any more transactions from the operational point of view. We decided to freeze the database in order to preserve it.

In figure 2 it is presented a portion of code extracted from a DBML document produced by a prototype used in the case study.

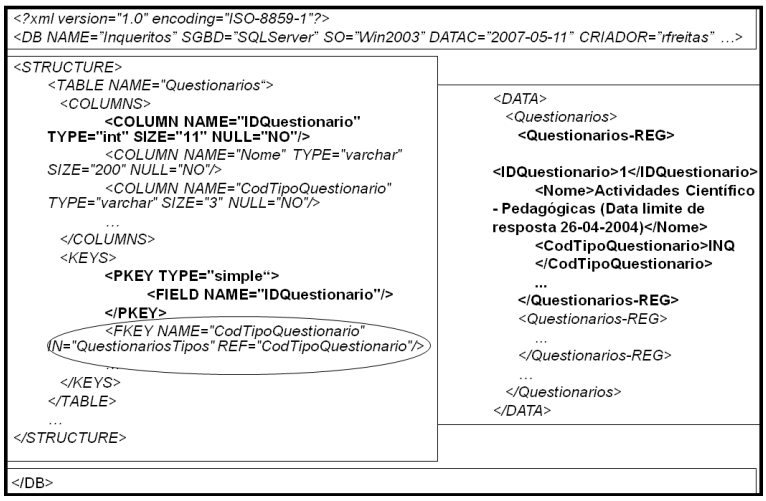


Fig. 2. DBML portion of the document – case study

After this brief example of the document and its format used to archive the database, we are able to analyze the archive focusing on the system architecture and the workflow of the database from its ingestion until its dissemination.

3.1 System Architecture

We will now seek to describe and analyze the architecture of the implemented system. The prototype is based on a web application with multiple interfaces. These interfaces have the mission to take a certain database and ingest it into the archive. The access to the archive in order to do all the necessary interventions on the system will also be done through those web interfaces.

Conceptually, the prototype is based on the OAIS [4] reference model. The OAIS model of reference does not impose rigidity with regard to implementation, rather it defines a series of recommendations.

The OAIS model of reference is concerned about a number of issues related to digital preservation: the process of information Ingestion into the system, the information storage as well as its administration and preservation, and finally information access and dissemination [6] [11].

However, the OAIS model does not impose any computer platforms, development language, database management systems (DBMS), interfaces, i.e., does

not condition the development of the system at the technological level involved. Instead, the model acts as a guide for those who wish to develop digital archives [4]. Figure 3 shows a conceptual design of the OAIS reference model.

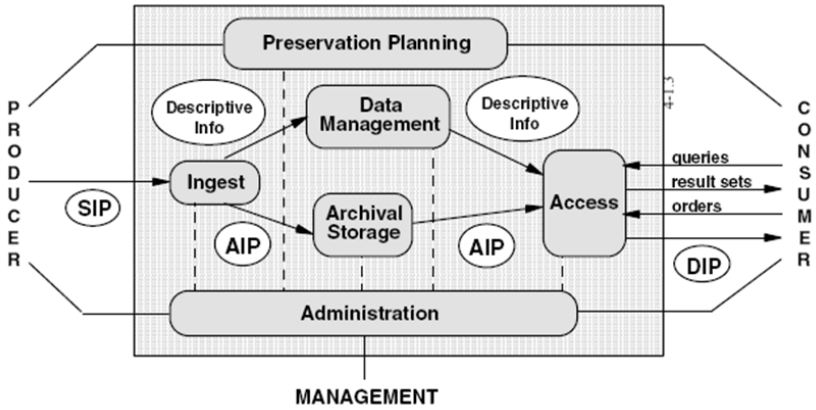


Fig. 3. OAIS Functional Entities. (Courtesy of Consultative Committee for Space Data Systems. "Reference Model for an Open Archival Information System (OAIS) – Blue Book," National Aeronautics and Space Administration, Washington, 2002) [4]

Three information packages are the base of the archival process: Submission Information Package (SIP), Archival Information Package (AIP) and Dissemination Information Package (DIP). Before ingestion begins, it is necessary to establish a Submission Agreement between the Producer and the Archive. Before dissemination starts, an Order Agreement between the Archive and the Consumer is established. Through these agreements both SIP and DIP constitutions are defined well as the specifications of the sessions for data submission and dissemination. The Administration component is responsible to define the AIP constitution – package that will be stored.

The SIP is composed by both descriptive and technical metadata and the digital content itself. The ingestion process includes the SIP validation delivered by the Producer. If the minimum requirements are achieved the package is ready to be archived. After this, the package is transformed into an AIP. At the other end of the archive the Consumer may query the OAIS trying to find the desired data. When the information is found the Consumer will issue a request to the system that will respond with DIPs – packages used in the dissemination process.

Inside the archive, the administration component manages the AIPs and participates in the ingestion and dissemination process. The preservation component is responsible for implementing preservation policies. Our prototype follows this conceptual model.

The Prototype implementation was a crucial phase of this work. We intend to implement a system capable of ingesting databases, in the form of information packages (DBML + metadata), for preservation. The developed system is based on a Web application and has multiple interfaces that allows not only the ingestion of information, but also its administration, preservation and dissemination. Figure 4 gives an idea of the preservation process over the prototype.

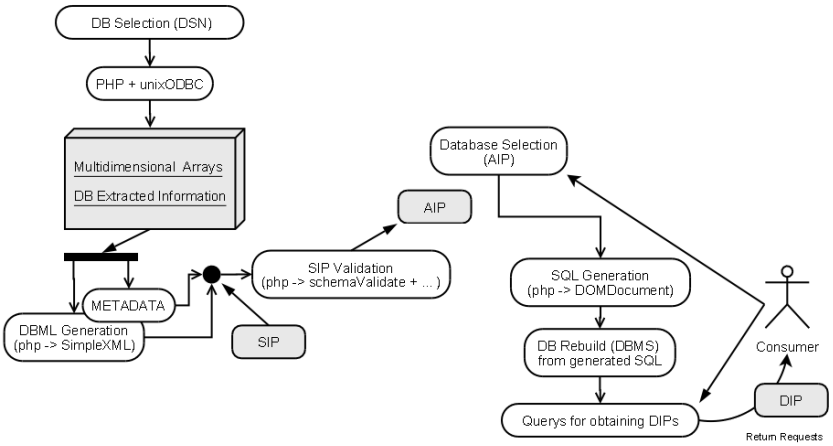


Fig. 4. Database preservation workflow

The several Web interfaces can only be accessible through a previous authentication on the system. The administration component manages these requests, and the various privileges with regard to the handling of information in the archive. The users of the system can be divided into two types, namely two profiles of users: administrators and users. A user has at his disposal the following list of operations:

- Creation of SIPs
- Ingestion of SIPs in the repository (AIPs)
- Consultation of the state of the repository
- Production of SQL [3] from AIPs
- Dissemination of DIPs

The administrator has at its disposal all operations available to users with the addition of operations associated with administration tasks:

- Management of users of the system
- Direct access to the file system (Repository directory)
- Manipulation of drivers (unixODBC) for connection to databases
- Monitoring of the state of obsolescence of information in the repository

– Actions of migration and refreshment of the stored information whenever necessary (preservation policies)

The practical implementation has proven to be very significant because it was possible to obtain interesting results. After the initial assembly of all the tools necessary to perform the tasks required, we quickly started to develop the prototype. Some adjustments were made to obtain better performance of the system.

It is important to refer that this work aimed to test the feasibility of relational database digital preservation using this approach. This was indeed possible, i.e., the objective of converting relational databases (different DBMS) into DBML was achieved. We were also able to rebuild the database in a DBMS from the DBML document in order to achieve the database dissemination.

4 Conclusion and Future Work

Different strategies (refreshment, migration and normalization) were combined to pursue the goal of digital preservation. The main strategy used is migration such as in the European project PLANETS. These strategies were used according to the class of digital objects that we addressed – Relational Databases. If the goal was the implementation of a repository for other family of digital objects the strategies may differ [7] [19].

Considering the properties (significant properties) of the database that we establish for preservation, the presented solution provides good results. However, in future work, we should be able to establish a consensus on several issues. Some of them are:

– walk further to determine the significant properties of a relational database and for each one of them or globally define the strategy that should be adopted;
– compare alternative strategies;
– how is it possible to ensure authenticity?
– how can we ensure preservation during the lifecycle of the database (while it evolves)?

We should study these issues by evaluating if the significant properties of a database are well preserved. We will also need to test if a preservation framework offers the possibility to query different versions of the database. Another important issue is, will the preservation framework be able to deal with hundreds of archived databases containing terabytes of data? During the research and framework improvement, we should be able to redefine concepts, if needed, and contribute effectively to the preservation of relational databases.

In conclusion, we can say that digital preservation is essential to ensure a future access to digital information legacy. There is no solution to completely solve this problem, and we do not know if it will ever occur. However, the fact that this issue has become the central subject of a scientific study, will probably contribute to solve the problem in the future.

References

1. Ronald Bourret, "XML and Databases, " Copyright 1999-2005 by Ronald Bourret. Last updated September, 2005
2. Lee Buck. "Data models as an XML Schema development method", XML 99, Phyladelphia, Dec. 1999.
3. Donald D. Chamberlin, Raymond F. Boyce, "SEQUEL: A Structured English Query Language," IBM, 1970
4. Consultative Committee for Space Data Systems. "Reference Model for an Open Archival Information System (OAIS) - Blue Book," National Aeronautics and Space Administration, Washington, 2002.
5. Edgar Codd, "A Relational Model of Data for Large Shared Data Banks," in Communications of the ACM, 1970.
6. Michael Day, "The OAIS Reference Model," Digital Curation Centre UKOLN, University of Bath, 2006
7. Claire Eager, "The State of Preservation Metadata Practices in North Carolina Repositories," Chapel Hill, North Carolina, 2003
8. Miguel Ferreira, "Introdução à preservacao digital - Conceitos, estratégias e actuais consensos," Escola de Engenharia da Universidade do Minho, Guimarães, Portugal, 2006.
9. Ricardo Freitas, "Preservação Digital de Bases de Dados Relacionais," Escola de Engenharia, Universidade do Minho, Portugal, 2008
10. M. Jacinto, G. Librelotto, J. Ramalho, P. Henriques, "Bidirectional Conversion between Documents and Relational Data Bases," 7th International Conference on CSCW in Design, Rio de Janeiro, Brasil, 2002.
11. B. F. Lavoie, "The Open Archival Information System Reference Model: Introductory Guide," Digital Preservation Coalition, Dublin, USA, Technology Watch Report Watch Series Report, 2004.
12. K.-H. Lee, O. Slattery, R. Lu, X. Tang and V. McCrary, "The State of the Art and Practice in Digital Preservation," Journal of Research of the National Institute of Standards and Technology, vol. 107, no. 1, pp. 93-106, 2002.
13. "PLANETS - Preservation and Long-term Access through NETworked Services" [Online]. Available: http://www.planets-project.eu/
14. J. Ramalho, M. Ferreira, R. Castro, L. Faria, F. Barbedo, L. Corujo, "XML e Preservação Digital," Dep. Informática, Universidade do Minho e Instituto dos Arquivos Nacionais, Torre do Tombo, 2007
15. J. Ramalho, M. Ferreira, L. Faria, R. Castro "Relational Database Preservation through XML modelling, " Extreme Markup Languages 2007, Montréal, Québec, 2007
16. J. Ramalho, P. Henriques, "XML and XSL - Da Teoria à Prática," FCA - Editora Informática, 2002.
17. "SIARD - Format Description," Swiss Federal Archives - SFA, 2008.
18. K. Thibodeau, "Overview of Technological Approaches to Digital Preservation and Challenges in Coming Years," presented at The State of Digital Preservation: An International Perspective, Washington D.C., 2002.
19. D. Waters, "Good Archives Make Good Scholars: Reflections on Recent Steps Toward the Archiving of Digital Information," 2002
20. C. Webb, "Guidelines for the Preservation of Digital Heritage," United Nations Educational Scientific and Cultural Organization - Information Society Division, 2003.

21. Wikipedia contributors, "Database models," in Wikipedia, The Free Encyclopedia, 2008. [Online]. Available: http://en.wikipedia.org/wiki/Database_models/
22. XML, "Extensible Markup Language", in W3C - The World Wide Web Consortium [Online]. Available: http://www.w3.org/XML/

Sharing botanical information using geospatial databases

João Silva, Cristina Ribeiro, and João Correia Lopes

Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias
s/n 4200-465 Porto, PORTUGAL
{ei04032,mcr,jlopes}@fe.up.pt

Abstract. Current botanical databases store taxonomic and specimen data, and can be queried using standard text-based queries. A simple example is retrieving all recorded specimens of *Metasequoia glyptostroboides*. In some botanical databases, this might result in a set of georeferenced results. However, they lack the ability to provide spatial queries, such as finding all occurrences of *Metasequoia* living in the FEUP campus area.
We propose a flexible platform for querying a spatially-enabled database. This platform can support a botanical database, extended with spatial querying capabilities. The details of the underlying database are hidden from its clients, but efficiency in handling spatial data and its spatial querying potential are essential. For full backwards compatibility, spatial queries can be blended with standard queries. This solution is technology-independent from the clients' point of view and uses the Darwin Core standard format for botanical information exchange.

1 The botanical domain

Botany is a large information domain, with over 287 000[1] different identified species, as of 2004. Also, Botany has a very structured model for representing information. Since Carl Linnaeus invented *Linnaean Taxonomy* [2], this model has been used as a standard to classify living organisms into categories, called *taxons*. As an example, the taxonomic classification of the olive tree is shown in Table 1.

1.1 Why does Botany need GIS?

The basis of most botanical databases has been the taxonomical classification. Large online databases like the PLANTS database [3], the Harvard University Herbarium (HUH) [5] or the Global Biodiversity Information Facility (GBIF) [1] are widely known and used. These databases include a basic way of associating

[1] in http://en.wikipedia.org/wiki/Plant.

Cladus (general)	Eukaryota
Regnum	Plantae
Divisio	Magnoliophyta
Classis	Magnoliopsida
Familia	Oleaceae
Genus	Olea
Species	Olea europaea

Fig. 1. Taxonomic classification for the olive tree

locality elements. The person who identifies an occurrence can append written references, such as "On Van Stadens Berg, nearest to Galgebosch"[2].

GBIF [1] takes the next step towards georeferencing. It offers a very large volume of georeferenced information, regarding not just plants. This information, however, comprises mostly geographically approximated occurrences, retrieved from historical records. A similar approach has been used in geographical systems dealing with forests [17].

Geospatial information is very important for botanists, because it allows them to trace the occurrences of specimens much more precisely, and match them to other geospatially-oriented phenomena, such as weather patterns in a specific area. This kind of information can be valuable in establishing cause-effect relationships between phenomena and provide insight on the evolution of endangered species or invasive plants, for example.

1.2 Botanical information exchange - Darwin Core

Since a lot of botanical information has already been catalogued in information systems, a new-born system for this domain must be able to interact with currently existing ones. Interoperablity is therefore one of the challenges of building online services to support biodiversity mapping [15]. For a system to be interoperable, it must use a standard information exchange format. XML (eXtensible Markup Language) is a widely used format for information exchange, being especially adequate for data transfer throughout the Web. Botanical data is no exception, and there is a proposed standard for this kind of information, *Darwin Core*, currently pending final approval by TDWG [4]. While not being (yet) a *de-facto* standard for biological information exchange, *Darwin Core* is already extensively used. Some good examples are several large projects in the biological domain[3].

[2] From the HUH [5] database, for the specimen with Collection Number 4697.

[3] Darwin Core, version 1.21 schema (revised version) is used in GBIF, MaNIS, HerpNet, OrNIS, and FishNet2 databases [10].

2 Geo-Botanical Repositories

There are several excellent public botanical databases. However, their query capabilities are built on top of text parameters. When a user queries the database, the parameter can be the required value for any level of taxonomic information or the *Collection Code*, for example. In a more *pseudo-spatial* approach, querying about the specimens in a specific country can yield georeferenced results, such as in the GBIF database [1]. However, only the results are spatial (points and place marks), not the queries themselves, e.g., users can specify values for certain text-based parameters (such as the botanical occurrence's country of origin), but cannot use arbitrary polygons as part of the query's restrictions.

2.1 What is right about current solutions

Several requirements for botanical databases have already been met. These proven approaches serve both as directions and examples.

The sharing of information is the first strong point in a botanical database. The GBIF database [1] offers a range of Web Services [6] that allow other applications to interact with it. This is good because it allows the sharing of information in interchangeable formats, such as *Darwin Core*. Selecting the correct formats for geospatial data viewing is also very important. With the spreading of solutions such as Google Maps and Google Earth, end-users are becoming more accustomed to using geospatial data in their everyday life. GBIF [1] saw this as an opportunity and now offers part of their data in KML[4] format for public download and viewing. Another important feature of a botanical database is its openness. Botanical databases gather information from various parts of the globe. Local institutions catalogue their specimens and send in the information, which is made publicly available. To allow a database such as this to grow, it must provide users with a simple way to insert specimen information and to retrieve it.

2.2 What is missing in current solutions

There is much untapped potential for a different approach:

> What if the user wants to specify an arbitrary *polygon* and query about the specimens recorded *within*/*outside*/...[5] the polygon?

This kind of questions can only be answered by spatial querying, supported by a geospatial database. A simple example is now presented to demonstrate the potential and usefulness of this approach.

In Figure 2 we can see a map of the Iberian Peninsula. In this map, the banners are placemarks for some fictional botanical occurrences. It seems obvious

[4] Keyhole Markup Language.
[5] Or any OpenGIS Consortium [8] Specification spatial restriction operator.

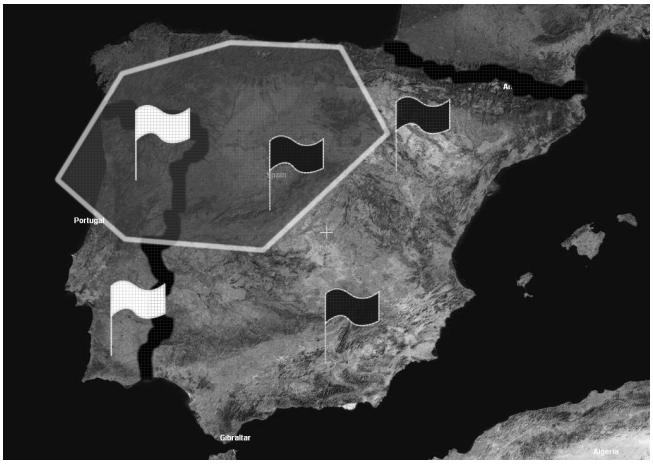


Fig. 2. The Iberian Peninsula, some botanical occurrences and a polygon

that botanical occurrences are not constrained by country borders, so it makes sense to search for occurrences in user-specified areas. It would be interesting for the user (person or machine) to use a polygon such as the one in Figure 2 as the query restriction, or "country border". In this case, the system would retrieve the two banners inside the polygon (a dark one and a light-coloured one).

This sort of *fine-grained* geospatial restrictions yields several interesting uses, such as:

- Finding patterns between botanical and weather-related phenomena in a given region;
- Tracing species propagation and identifying potential pests or invasive species;
- Providing information to tourists interested in visiting places where certain species of plants exist;
- Helping botanical parks to compose their daily tours using spatial queries. (see Section 6 for more details);
- Helping pharmaceutical companies keep trace of places where certain botanical species are discovered;
- Helping environmentalists keep trace of species evolution in a given area;
- Helping architects in charge of city planning/landscaping to keep track of the specimens planted in any city garden or park.

Each of these uses can translate into a client for the system, using its spatial querying capabilities to support the client's business logic.

3 A spatial querying system and its requirements

We propose a spatial query language aimed at botanical repositories. Its use will be illustrated in a botanical data server and some example clients.

3.1 The data model

The data model underlying the repository captures several concepts pertaining to the classification of species and the cataloguing of specimens, linking these to their geographical location. This model [18], includes classes for taxonomic classification, such as `Familia` or `Genus`. Physical locations are represented by `Geometries`. For specimen classification purposes there is an `Occurrence` class. Other classes such as `Tour` help in the management of a botanical website. More detail can be found in the full project report [16]. The data model includes the concepts available in the Darwin Core (DC) standard. This grants interoperability to the system, because it is able to easily import and export DC formatted data.

3.2 The query language

The query language, intended to access the information in the repository, was designed with the following features in mind:

1. **Simplicity**
   Hiding intricate low-level implementation details of spatial database access, as these are mostly irrelevant to end-users. As an example, to retrieve all specimens inside an area, the user only needs to know what a polygon *intersection* is.
2. **Flexibility**
   Allowing for elaborate or simple queries, depending on the end users' needs.
3. **Expressiveness**
   Users can access the whole set of features offered by the underlying database.
4. **Technology independence**
   Client programmers can access the central database, regardless of the programming language that they use or other technological constraints.
5. **Interoperability**
   Allowing the language to be easily used by machines.

To fulfill all these requirements, the *PlantGIS* query language was developed. *PlantGIS* takes its inspiration from the standard SQL notation, some *Darwin Core* attribute names and, of course, the *OpenGIS Consortium*(OGC) spatial restriction operators [8], such as `equals`, `disjoint`, `intersects`, `touches`, `crosses`, `within`, `contains` or `overlaps`.

The language is very simple, as can be seen in its graphical representation [20]. It resorts to the meaning of each of the spatial operators defined by OGC [8]. To compose the query in the example, the user has to know that the `within` operator will include all occurrences *spatially within* [8] the argument polygon. To use a more informal description, this operator will return polygons "fully inside" the argument polygon.

To provide a high degree of flexibility, the language supports the `AND`, `OR` and `NOT` boolean operators. The precedence is given by the level of each query

in the XML tree. Common *text-based* querying capabilities widely used in other databases are also available here. Both types of queries can be combined freely, as demonstrated in the example query [19].

Using XML allows interaction with a broad range of clients. From handheld terminals to full-fledged Web servers, there is almost always an XML parsing library available. Using XML transported via Web Services, technology restrictions are dramatically reduced.

3.3 Information output formats

Query results are produced in two main formats: *Darwin Core* and KML. KML is an interesting format for exchanging spatial information. It is Google Earth and Google Maps' default document format. Some botanical databases also offer their results in KML format. GBIF, for example, uses it as its geospatial information exchange format.

4 A botanical data server

All botanical databases have their specific needs; however, all of them share the need for geospatial information and querying. With this in mind, the notion of a central server application to serve only the clients' common needs becomes an interesting alternative. The specific needs of each domain are left for clients to implement. The database would then grow by exchanging its geospatial querying capabilities for some information about the clients' specimens.

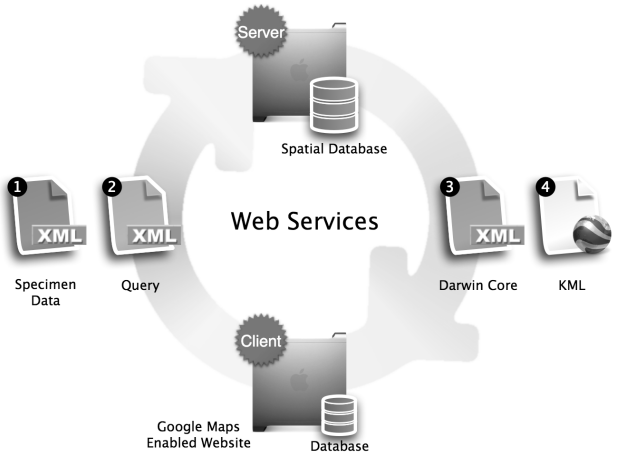


Fig. 3. Client/Server interaction

Figure 3 is a high level physical diagram, depicting the interactions between the server and its client applications.

Clients can send *specimen occurrence* data to the server, to publish new botanical occurrences. The server processes the request and responds to the client with a global identifier. This identifier can, from then on, be used by the client to have direct access to that occurrence's record in the server's spatial database.

Queries, formatted in *PlantGIS*, are sent to the server. They are validated against the *PlantGIS* query language specification. As for the results of the query, they represent specimen occurrences (points or polygons). The available formats for output are *Darwin Core* and KML. *Darwin Core* output ensures the interoperability of this system, and KML is used to improve result visualisation by humans. Clients can include Google Maps visualizers for query results received from the server. At the same time, KML files can also be opened in Google Earth for offline viewing.

4.1 Client scenario

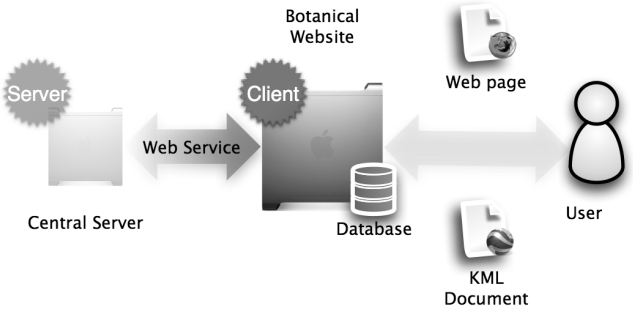


Fig. 4. Physical diagram for a client scenario

An hypothetical client scenario is shown in Figure 4. In this case, the client is a website, with its own database and business logic. The user interacts with the client, which in turn uses the server's spatial querying capabilities to provide users the information they need.

5 Technologies

The server application was designed to be an API. As such, it must be flexible, open, and easily expandable. Thus, any of the technologies selected had to be based on *open-source* software.

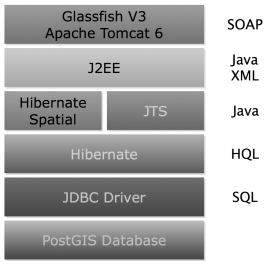


Fig. 5. Technologies used in the server application

Figure 5 shows the choice of technologies for this system. To the right of each layer is the predominant language/information exchange format it handles.

PostGIS Database PostGIS is an geospatial extension for the PostgreSQL DBMS. This extension adds dedicated spatial data types, such as `Geometry`, and spatial operators from the OpenGIS Consortium. The most prominent *open-source* alternative to PostGIS is MySQL with Geospatial Extensions. However, one feature sets the extension of PostgreSQL apart from MySQL with Geospatial Extensions: its full OpenGIS Consortium standards compliance [12,11].

Hibernate Hibernate is a widely used object-relational mapping solution for Java. It makes database interaction transparent to the programmer, mapping each table in the database to Java classes. Its query language, HQL, is translated by the framework to SQL in run-time and injects it into the database. This introduces additional overhead in the system but allows for complete Object Oriented querying, since HQL can include references to Java object fields and not only table/column names. An equally important feature is database vendor independence. Since no *native* SQL is actually written, the database adaptor (JDBC Driver) can be changed if one decides to replace the underlying database. The Hibernate framework will just map its HQL to a different SQL dialect to encompass the change. Also, using Netbeans, database changes are quickly dealt with because the IDE allows for automatic regeneration of the corresponding Java objects, and since there is no hard-coded SQL, the IDE itself helps with the process of *refactoring*.

Hibernate Spatial Hibernate natively supports the most common database datatypes (`integer`, `varchar` and so on). Geospatial data types, such as `Geometry` are handled by Hibernate Spatial, allowing Hibernate to map these data types and making their usage transparent to the programmer. This plugin takes HQL with geospatial data types and generates the native SQL (depending on the underlying JDBC adapter) to take advantage of the indexes and optimization of spatially-enabled databases.

Java Topology Suite (JTS) Java Topology Suite is an efficient library for handling geometry-related operations. It provides methods for calculating `union`s, `intersection`s and other operations between `Point`s and `Polygon`s. Hibernate Spatial works in conjunction with this library to perform all the necessary calculations when querying the database using spatial restrictions[6] and return the records that match the criteria.

6 The *FEUP Park* Client

To evaluate the usefulness of the developed server application, a "test case" client application was developed. This application was named *FEUP Park* and simulates a small botanical park's most common use cases[7]. It demonstrates the usage of the server's spatial querying capabilities.

6.1 Insertion of new records

The client application is capable of inserting new occurrences. There are several interesting features in the web page designed for this purpose. Through real-time KML parsing, occurrence insertion is easier for the user. The system is able to receive KML files and show lists of the polygons and points contained in that file. These entities can then be selected by the user and allocated to the new occurrence. Also, the client can display uploaded KML files on Google Maps to assist the user in building the query. Figure 6 shows this visualizer; the two lists at the bottom show the names of points and polygons inside the uploaded KML file.

KML files can be generated from Google Earth (after manual tagging of points and polygons within the program) or even up/downloaded from GPS receivers. An example is the "GPS utility" software, that can be used with *Garmin* GPS receivers [13]. This makes geo-referencing easier, because users can walk around the park, tag the specimens' locations and save them to the GPS receiver. Then, using this Web interface, they can upload the generated KML and tag each point with its associated taxonomic classification.

Using data collected from the *Wikispecies* website, the system can fill in the upper taxons' values after selecting a value for a taxon. This is done in real time, using AJAX (Asynchronous Javascript and XML). Upon the submission of an occurrence, data is stored locally in the park's database and also sent to the central server.

6.2 Spatial Querying

There is a dedicated area for querying the specimen database, assisting users in building their queries.

[6] OpenGIS Consortium specification [8] restrictions such as `within` or `intersects`.

[7] This application is currently under development and a demonstration version will soon be available at `http://paginas.fe.up.pt/~ei04032/plantgis`.

Fig. 6. The client's Google Maps visualizer.

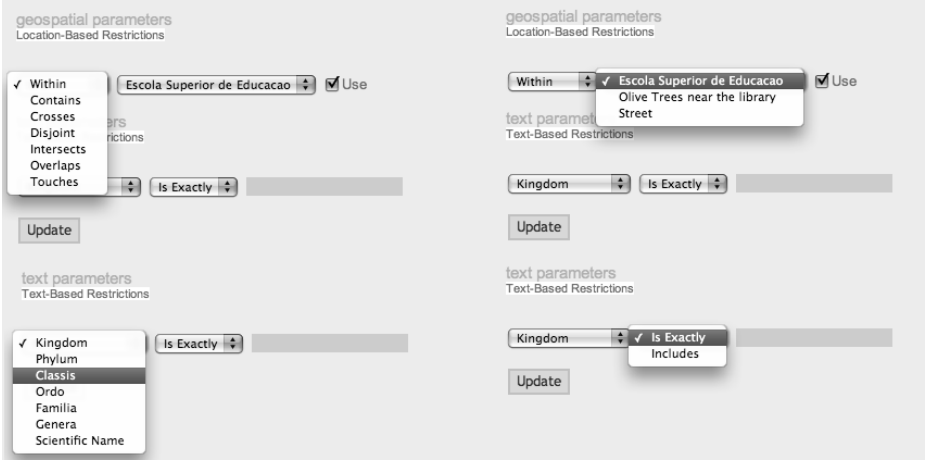


Fig. 7. Selecting operators and polygons while building a query

Figure 7 shows part of the query building process, with all the parameters and operators available in the client. The user can select several of these. The client translates their options to a *PlantGIS* query, sends it to the server, and retrieves the results in Darwin Core or KML. Here again, if the user has uploaded his/her own KML files, polygons inside them can be used as query arguments.

Designing Tours The Park Client interacts with the server but has its own requirements. To illustrate the support that the server can lend to such activities, a simple tour making page was developed. It is built on top of the query page. The user can order the results of a query (occurrences) and save a new tour. It then becomes available for viewing on Google Maps or Google Earth (KML exportation).

7 Conclusions

This proposal aims at enabling botanical databases with geospatial querying capabilities. To achieve that goal, it offers a query language for geo-botanical data. The proposed system is also able to exchange biological information with similar databases using standard formats. This interoperable, centralized system can be used by many types of clients, and some examples have been identified. One of these clients has also been demonstrated, showing some of the usages of the proposed solution.

8 Future Work

Some future development perspectives for this system include a client for mobile devices. This client will be capable of reading RFID (Radio-Frequency Identification) tags from botanical specimens in the field, retrieve their information and display it on the mobile terminal's screen. The portable terminals could be used by park visitors, replacing conventional tags as the main identification method of specimens. An experiment has already taken place in the context of this work, in a collaborative effort between FEUP and iUZ Technologies[8].

Also, a more sophisticated client for building queries could be developed, enabling users to specify more complex queries than those currently supported by the demonstration web application.

References

1. Global Biodiversity Information Facility: GBIF Portal. `http://data.gbif.org/welcome.htm` (Consulted on May 2009)

[8] A small video was shot in the FEUP campus area, depicting the usage of this prototype client. It can be seen at `http://paginas.fe.up.pt/~ei04032/plantgis/paper/DemoRFID.mov`.

2. Wikimedia: Linnaean Taxonomy. http://en.wikipedia.org/wiki/Linnaean_taxonomy - (Consulted on May 2009)
3. USDA, NRCS. 2009 : The PLANTS Database. http://plants.usda.gov. National Plant Data Center, Baton Rouge, LA 70874-4490 USA. - (Consulted on May 2009)
4. Biodiversity Information Standards: TDWG: Homepage. http://www.tdwg.org/ - (Consulted on May 2009)
5. President and Fellows of Harvard College : Harvard University Herbarium. http://asaweb.huh.harvard.edu:8080/databases/specimen_index.html - (Consulted on May 2009)
6. GBIF: About - Using data from the GBIF Portal. http://data.gbif.org/tutorial/services - (Consulted on May 2009)
7. GBIF: Datasets - A - GBIF Portal http://data.gbif.org/datasets/ - (Consulted on May 2009)
8. Open Geospatial Consortium Inc: OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture. - 2006-10-05
9. Biodiversity Information Standards: Purpose and design goals of the Darwin Core. http://wiki.tdwg.org/twiki/bin/view/DarwinCore/GeospatialExtension - (Consulted on May 2009)
10. Biodiversity Information Standards: Darwin Core Versions. http://wiki.tdwg.org/twiki/bin/view/DarwinCore/DarwinCoreVersions - (Consulted on May 2009)
11. MySQL AB, Sun Microsystems, Inc.: MySQL 5.0 Reference Manual http://dev.mysql.com/doc/refman/5.0/en/mysql-gis-conformance-and-compatibility.html - (Consulted on May 2009)
12. PostgreSQL Global Development Group: PostGIS receives OGC compliance http://postgis.refractions.net/pipermail/postgis-users/2006-September/013058.html - (Consulted on May 2009)
13. GPS Utility Ltd.: GPS Utility - Receivers http://www.gpsu.co.uk/gpsrecs.html - (Consulted on May 2009)
14. Network Working Group: RFC 2045 - Multipurpose Internet Mail Extensions (MIME) - Part One: Format of Internet Message Bodies http://tools.ietf.org/html/rfc2045 - (Consulted on May 2009)
15. Robert Guralnick and David Neufeld: Challenges building online GIS services to support global biodiversity mapping and analysis: Lessons from the mountain and plains database University of Colorado Museum, Department of Ecology and Evolutionary Biology Biodiversity Informatics - pp. 56-69 - February 2005
16. João Silva: Master's Thesis, Sharing botanical information using geospatial databases Faculdade de Engenharia da Universidade do Porto - Available July 2009
17. Paulo Costa de Oliveira Filho, Atilio Antonio Disperati et al. Um sistema de informacoes geograficas como suporte a um experimento florestal na flona de Irati-PR Unicentro, Depto. de Engenharia Ambiental PR 153, km 7 Bairro: Riozinho - 84.500 - Irati - Paraná - April 2005 (Article)
18. João Miguel Rocha da Silva Data Model http://paginas.fe.up.pt/~ei04032/plantgis/paper/data_model.pdf - May 2009
19. João Miguel Rocha da Silva An example of a query, formatted in PlantGIS http://paginas.fe.up.pt/~ei04032/plantgis/paper/Sample.xml - May 2009
20. João Miguel Rocha da Silva PlantGIS Language graphical representation http://paginas.fe.up.pt/~ei04032/plantgis/paper/pgis_tree.jpg - May 2009

Sessão 4B: Computação Distribuída e de Larga Escala

Dependability in Aggregation by Averaging

Paulo Jesus, Carlos Baquero, and Paulo Sérgio Almeida

University of Minho (CCTC-DI)
Campus de Gualtar, 4710-057 Braga, Portugal
{pcoj, cbm, psa}@di.uminho.pt

Abstract. Aggregation is an important building block of modern distributed applications, allowing the determination of meaningful properties (e.g. network size, total storage capacity, average load, majorities, etc.) that are used to direct the execution of the system. In the recent years, several approaches have been proposed to compute aggregation functions on distributed settings, exhibiting different characteristics, in terms of accuracy, time and communication complexity. However, the majority of the existing aggregation algorithms exhibit relevant dependability issues, when prospecting their use in real application environments. In this paper, we reveal some dependability issues of aggregation algorithms based on iterative averaging techniques, giving some directions to solve them. This class of algorithms is considered robust (when compared to common tree-based approaches), being independent from the used routing topology and providing an aggregation result at all nodes. However, their robustness is strongly challenged and their correctness often compromised, when changing the assumptions of their working environment to more realistic ones. The correctness of this class of algorithms relies on the maintenance of a fundamental invariant, commonly designated as *mass conservation*. We will argue that this main invariant is often broken in practical settings, and that additional mechanisms and modifications are required to maintain it, incurring in some degradation of the algorithms performance. In particular, we discuss the behavior of three representative algorithms (*Push-Sum Protocol* [1], *Push-Pull Gossip protocol* [2] and *Distributed Random Grouping* [3]) under asynchronous and faulty (with message loss and node crashes) environments. More specifically, we propose and evaluate two new versions of the *Push-Pull Gossip protocol*, which solve its message interleaving problem (evidenced even in a synchronous operation mode).

1 Introduction

With the advent of multi-hop ad-hoc networks, sensor networks and large scale overlay networks, there is a demand for tools that can abstract meaningful system properties from given assemblies of nodes. In such settings, aggregation plays an essential role in the design of distributed applications [4], allowing the determination of network-wide properties like network size, total storage capacity, average load, and majorities. Although apparently simple, in practice aggregation has reveled itself to be a non trivial problem, especially when seeking

solutions in distributed settings, where no single element holds a global view of the whole system.

In the recent years, several algorithms have addressed the problem from diverse approaches, exhibiting different characteristics in terms of accuracy, time and communication tradeoffs. A useful class of aggregation algorithms is based on *averaging* techniques. Such algorithms start from a set of input values spread across the network nodes, and iteratively average their values with active neighbors. Eventually all nodes will converge to the same value and can estimate some useful metric. Averaging techniques allow the derivation of different aggregation functions besides average (like count, sum, maximum and minimum), according to the initial combinations of input values. E.g., if one node starts with input 1 and all other nodes with input 0, eventually all nodes will end up with the same average $1/n$ and the network size n can be directly estimated by all of them [5].

The main objective of this work is to expose relevant dependability issues of existing aggregation by averaging algorithms, when challenged by practical implementations in realistic scenarios. For this purpose, we discuss and evaluate the behavior of three representative averaging algorithms, when confronted with practical concerns like communication asynchrony, message loss and node failure. We choose to analyze the following algorithms: *Push-Sum Protocol* [1] (PSP), *Push-Pull Gossip protocol* [2] (PPG), and *Distributed Random Grouping* [3] (DRG). To the best of our knowledge this is the first evaluation of averaging algorithms focusing on dependability and taking into account practical implementation concerns.

The remaining of this paper is organized as follows. We briefly refer to the related work on aggregation algorithms in Section 2. A detailed analysis of some representative averaging aggregation algorithms, concerning their practical implementation on real distributed systems, is discussed in Section 3. In Section 4, we propose two solutions to fix the interleaving issues exhibited by PPG, and compare them with the original algorithm in a common simulation environment. Finally, we make some concluding remarks in Section 5.

2 Related Work

Several aggregation algorithms have been proposed in the last years, tackling the problem for different settings, and yielding different characteristics in terms of accuracy, time and communication complexity.

Classical approaches, like TAG [6], perform a tree-based aggregation where partial aggregates are successively computed from child nodes to their parents until the root of the aggregation tree is reached (requiring the existence of a specific routing topology). This kind of aggregation technique is often applied in practice to Wireless Sensor Network (WSN) [7]. Other tree-based aggregation approaches can be found in [8], and [9]. We should point out that, although being energy-efficient, the reliability of these approaches may be strongly affected by the inherent presence of single-points of failure in the aggregation structure.

Another common class of distributed aggregation algorithms is based on averaging techniques [1, 2, 5, 3, 10]; Here, the values of a variable across all nodes are averaged iteratively. This kind of approaches is independent from the routing topology, often using a gossip-based communication scheme between peers. In this study, we will specifically discuss three of these approaches: PSP [1], PPG [2], and DRG [3].

Alternative aggregation algorithms based on the application of *probabilistic* methods, can also be found in the literature. This is the case of Extrema Propagation [11] and COMP [12], which reduce the computation of an aggregation function to the determination of the minimum/maximum of a collection of random numbers. These two techniques tend to emphasize speed, being less accurate than averaging approaches.

Specialized probabilistic algorithms can also be used to compute specific aggregation functions, such as COUNT (e.g. to determine the network size). This type of algorithms essentially relies on the results from a sampling process to produce an estimation of the aggregate, using properties of random walks, capture-recapture methods and other statistic tools [13–16].

3 Analysis

We analyze the practical implementation of aggregation algorithms based on averaging techniques, envisioning their deployment on real distributed network systems (e.g WSN and P2P overlay networks). In particular, we discuss three representative algorithms from this class: PSP [1], PPG [2], and DRG [3]. The performed analysis focuses on the reliability of these algorithms, in order to provide an accurate aggregation estimate, on realistic application scenarios, which are commonly governed by communication asynchrony, and failures. More specifically, this analysis is confined to four main practical settings/concerns:

1. **Synchronous model** – refers to the common synchronous operation mode, where the algorithms execution proceeds in lock-step rounds and without faults. In practice, where networks are typically asynchronous, it is possible to implement synchrony over an asynchronous fault-free network (see Chapter 16 of [17]), using a *synchronizer*. Notice that even under this strong synchrony assumption, algorithms that span more than one round, may see their messages interleaved across rounds;
2. **Asynchronous model** – in these settings, message transit takes a finite but unknown time. We consider that nodes communicate using FIFO channels, and no faults will occur. This means that no message in transit can be surpassed by any other message from the same source. Commonly, in practice, the transport communication layer (e.g. TCP) can provide a reliable and ordered message delivery, retransmitting undelivered data packets and using a sequence number to identify their order. Like in previous settings, interleavings may also frequently occur;
3. **Message loss** – corresponds to the loss of communication data, due to a temporary link or node failure. In this settings, we consider asynchronous

non-FIFO communication, where no guarantee on message delivery is made (like in UDP);
4. **Node Crash** – refers to the permanent failure of a node at an arbitrary time – *crash-stop* model. If a node crashes, it will no longer receive nor send messages, and will be considered as permanently unavailable from that time on.

Aggregation algorithms based on averaging are independent from the network routing topology, and able to produce an estimate of the resulting aggregate at every network node. The main principle of this kind of algorithms is based on an iterative averaging process between small sets of nodes. Eventually, all nodes will converge to the correct value by performing the averaging process among all the network.

The correctness of averaging aggregation algorithms depends on the maintenance of a fundamental invariant, commonly designated as the "mass conservation". This property states that the sum of the aggregated values of all network nodes must remain constant along the algorithm's execution, in order for it to converge to the correct result [1].

This kind of aggregation technique intends to be more robust than classical tree-based approaches, by removing the dependency from a specific routing topology, and providing an estimation of the aggregate at every node. For instance, these algorithms are often based on a gossip (or epidemic) communication scheme, which is commonly thought to be robust. Although, similarly to gossip communication protocols [18], the robustness of aggregation algorithms can be challenged, according to the assumptions made on the environment in which they operate.

Next, we discuss and expose dependability issues of some aggregation algorithms operating on a fixed network, but under more realistic assumptions, such as: asynchronous message exchanges, link and node failures.

3.1 Push-Sum Protocol

The PSP [1] is a simple gossip-based aggregation algorithm, essentially consisting on the distribution of shares across the network. Each node maintains and iteratively propagates information of a *sum* and a *weight*, which are sent to randomly selected nodes. In more detail: at each time step t (synchronous round), each node i sends to a target node (chosen uniformly at random) and to itself, a pair $(\frac{1}{2}s_{t,i}, \frac{1}{2}w_{t,i})$ containing half of its current sum $s_{t,i}$ and weight $w_{t,i}$; the values $s_{t,i}$ and $w_{t,i}$ are updated by the sum of all contributions received in the previous time step $(t-1)$; an estimate of the aggregation result can be provided at all nodes by the ratio $s_{t,i}/w_{t,i}$.

Distinct aggregation functions can be computed with these scheme, by varying only on the starting values of the *sum* and *weight* variables at all nodes. For instance, considering an initial input value x_i at each node i, the following functions can be computed, resorting to distinct initializations: AVERAGE ($s_{0,i} = x_i$ and $w_{0,i} = 1$ for all nodes); SUM ($s_{0,i} = x_i$ for all nodes, only one node sets

$w_{0,i} = 1$ and the remaining $w_{0,i} = 0$); COUNT ($s_{0,i} = 1$ for all nodes, only one with $w_{0,i} = 1$ and the others $w_{0,i} = 0$).

The continuous execution of this protocol throughout all the network allows the eventual convergence of all nodes estimates to the correct aggregation value, as long as none of the exchanged values are lost. As stated by the authors, the correctness of the algorithm relies on the *mass conservation* property. In particular, when no messages are in transit, for any time step t, the value $\sum_{\forall i} \frac{s_{t,i}}{w_{t,i}}$ is constant.

Aware from the crucial importance of this invariant, Kempe et al. considered a variation of the algorithm to cope with message loss and the initial failure of some nodes[1]. They assume that all nodes possess the ability to detect when their messages did not reach their destination, and modified the algorithm to send the undelivered data to the node itself, in order to recover the lost "mass". Furthermore, they assume that a node can only orderly leave the network, after sending all sums and weights to its peers, not supporting node crashes. In light of these stated assumptions, we extend the discussion of PSP under more realistic settings.

Synchronous model The proposed algorithm guarantees the maintenance of the mass conservation invariant on a synchronous execution model without faults, assuring its correctness – convergence to the true aggregation result.

Asynchronous model No issue was identified under asynchronous settings. Message delays may reduce the convergence speed of the algorithm, but will not compromise its correctness (as long as no message is lost). If at some arbitrary point t we stop the execution of the algorithm and wait for all message to be received and processed, we can verify that the ratio $s_{t,i}/w_{t,i}$ will meet the mass conservation property (the value $\sum_{\forall i} \frac{s_{t,i}}{w_{t,i}}$ will be equal to the initial value $\sum_{\forall i} \frac{s_{0,i}}{w_{0,i}}$).

Message Loss As referred by the authors, in order to support message loss, independently from its cause (temporary link/node failure, or initial permanent failures that makes some nodes unreachable), they assume that each node is able to detect messages that have not reached their destination. In this case, the lost mass will be re-sent to the source node itself. This is a very unrealistic assumption. Using an acknowledgement-based scheme to infer message loss, as suggested, would amounts to solving the *coordinated attack problem*, which in an asynchronous model under possible message loss has been shown to be impossible [19]. Furthermore, even if it were possible, it would introduce additional waiting delays in the protocol, in order to receive a delivery notification for each sent message.

[1] *Initial failure*, refers to nodes that have failed from the beginning of the computation.

Node Crash This algorithm does not support node crash. In order to maintain the correctness of the aggregation process, nodes will have to leave the network neatly, after sending all their mass to another node. Such optimistic assumption cannot be often used in practice, since node failures are not likely to be predicted. Nevertheless, one could consider a mechanism in which all the nodes state will be consistently replicated (at each neighbor), enabling alive nodes to subsequently recover the "mass" from a crashed node.

In order to be robust against node failures, G-GAP [10] extended the PSP, implementing a scheme based on the computation of recovery shares and the explicit acknowledgement of mass exchanges between peers. However, this approach provides only a partial support against this type of faults, supporting discontiguous node crashes (assuming that two adjacent nodes do not fail within a short time period of each other).

3.2 Push-Pull Gossip

The PPG protocol described in [5] and [2] is based on an anti-entropy aggregation process, being quite similar to PSP. The main difference of this algorithm relies on its *push-pull* process, which enforces a symmetric pairwise mass exchange between peers. The induction of this action/reaction pattern, coerces the average settlement between pairs, intending to immediately resolve the differences between nodes. The iterative execution of this push-pull process across all the network will provide the convergence of the algorithm to the correct value (faster, when compared to PSP).

Periodically, each node sends its current aggregation value to another node chosen uniformly at random – *push*, and waits for the response with the value from the target node – *pull* – to further apply the aggregation function, obtaining a new estimate of the aggregate. Each time a node receives a push message, it sends back its current value, and only then computes the new aggregate, averaging the received and sent value. At the end of a push-pull operation, both involved nodes will yield the same result.

In [5] the authors do not address mass conservation issues (like the impact of link/node failures), focusing on the efficient implementation of a pair selection method, and considering the extension of the algorithm with a restarting mechanism (executing the protocol during a predefined number of cycles, depending on the desired accuracy) in order to be adaptive and handle network changes (nodes joining/leaving). The subsequent study in [2] proposed a solution for the gossip-based aggregation directed to its practical applicability, considering some modifications to cover some practical issues, like: using timeouts to detect possible faults, ignoring the data exchanges in those situations; executing several instances of the algorithm in parallel to reduce the effect of message loss; making use of a restart mechanism to cope with node crashes.

We should point out that along the discussion carried out by the authors, they intrinsically assume that the core of the push-pull process is atomic, also referred as "the variance reduction step" ($w_i = w_j = (w_i + w_j)/2$). In practice,

additional modifications must be considered to guarantee the atomicity of *push-pull*. Like common aggregation algorithms by averaging, the correctness of PPG depends from the maintenance of the mass conservation property. Due to its core resemblance with PSP, one could expect to find similar practical issues when prospecting its use in real application scenarios. However, the push-pull process will introduce additional atomicity constraints, that will restrain the use of this algorithm even under a synchronous operation mode.

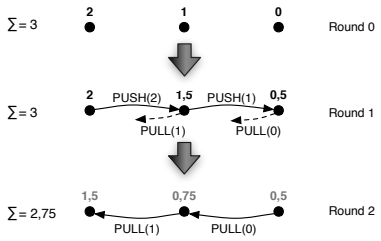


Fig. 1. Violation of the mass conservation invariant in the Push-Pull Gossip protocol, due to message interleaving (approximation values at the end of each round).

Synchronous model Even in synchronous settings (lock-step execution), the message delivery order may affect the convergence to the true result of PPG. It is fundamental to guarantee the atomicity of the push-pull process in those settings, in order to conserve the mass of the system. If a node starts a push-pull and receives a push message from a third-party node in the middle of the process (after sending the push and before receiving the pull message), then it will first update its approximation value according to the third-party data, before receiving the pull message, and use the result to update its value again with the pull data. This occurrence will introduce an asymmetry in the results produced from the push-pull (different values may be computed by each element), which will incur in a violation of the mass conservation, depending on the involved values (greater as greater is the discrepancy between them). Thus, the correctness of the algorithm is affected by message interleaving, as depicted in Figure 1. We will detail this issue in Section 4, proposing some modifications to the algorithm to guarantee the push-pull atomicity.

Asynchronous model The interleaving problem of PPG will also be found in asynchronous settings. Moreover, message delays will increase the possibility to interfere in an ongoing push-pull operation, and consequently break the mass invariant as previously described.

Message Loss In practice, considering the independent failure of a push and pull message, the loss of each one of those messages will yield different effects on

the PPG protocol: if the push message is lost, then no node will update its state (the source will timeout and the target will not receive the message) and the mass of the system will be conserved, only slowing down the algorithm convergence; On the other hand, if the pull message is lost (meaning that the push message was successfully received), then only the target node of the exchange process will update its value and the source will timeout waiting for a response, creating an undesirable asymmetry in the process and a consequent violation of the mass conservation invariant.

The authors do not make any assumption regarding the detection of message loss, neither concerning the restoration of the system mass. Instead of that, they consider the concurrent execution of multiple instances of the algorithm, discarding the lower and higher estimates and reporting the average of the remaining executions as the result. This solution does not solve the mass conservation issue due to message loss, and only reduces its impact in the quality of the produced estimate. Further considerations must be made to ensure the algorithm's correctness in this settings.

Node Crash The removal of nodes from the system originates a violation of the mass conservation property. In this case, the authors did not show any concern about recovering the mass changes provoked by unpredictable node crashes (not distinguishing crashes from nodes voluntarily leaving the network), but recognize the derivation of a subsequent estimation error. In particular, they specifically characterize the error of the average estimation as a function of the constant proportion of nodes failing (at each cycle), considering that nodes crash before each cycle, and intrinsically assuming (once more) that each variance reduction step is atomic (which is unreasonable in practice).

Indubitably, the introduced estimation error is provoked by the loss of the values held by crashed nodes, and leads to the convergence of the algorithm to an incorrect aggregation result. The authors consider the periodic restart of the PPG algorithm to cope with this issue, reinitializing the execution of the algorithm with clean input values (restoring the correct mass of the system). To detect the possible failure of nodes, they consider the use of a timeout, skipping the exchange step when the timeout expires. Hopefully, the restart mechanism will minimize the effect of the introduced convergence error, being transient to each epoch. Nevertheless, this issue invalidates the continuous use of the algorithm (without restarting), and may incur in unpredictable approximation results, since the correctness of the algorithm will be broken.

3.3 Distributed Random Grouping

A distinct approach based on a Distributed Random Grouping (DRG) is proposed in [3]. Unlike previous gossip-based aggregation algorithms, DRG was designed to take advantage of the broadcast nature of wireless transmissions (where all nodes within radio range will be prone to hear a transmission), directing its use for WSN. In a nutshell, the algorithm essentially consists on the continuous

creation of random groups across the network, to successively perform in-group aggregations. Over time, ensuring that the created groups overlap, the estimated values at all nodes will eventually converge to the correct network-wide aggregation result.

DRG defines three different working modes that coordinate its execution: *leader*, *member*, and *idle*. The algorithm operates as follows: according to a predefined probability, each node in idle mode can independently decide to become leader, and broadcast a Group Call Message (GCM), subsequently waiting for joining members; all (remaining) idle nodes respond only to the first received GCM with a Joining Acknowledgment (JACK) tagged with their aggregate estimation, changing their mode to become members of that group; after gathering the estimate values of group members from all received JACKs, the leader computes the new group aggregate and broadcasts the result in a Group Assignment Message (GAM), returning to idle mode and setting its own estimate with the newly calculated value; each member of a group waits for the GAM from its leader to update its local estimate with the result within, not responding to any other request until then, and returning to idle mode afterwards.

The performance of this algorithm is highly influenced by its capacity to create overlapping aggregation groups (size and quantity of groups), which is defined by the probability of a node to become leader. In terms of practical concerns, the authors make some considerations about termination detection for the algorithm, and consider the occurrence of message collisions and link failures, but only analyze their effect at the initial stage of the group creation (at the GCM level). In particular, they assume that link failures only happen between grouping time slots, which is an unrealistic assumption. A thorough analysis of these issues should be performed across the algorithm, considering also their impact at the JACK and GAM level.

Synchronous model DRG will work in synchronous settings, as long as each node is still only able to join at most one group at each time, which guarantees the maintenance of the mass conservation property.

Asynchronous model In asynchronous settings, beside the exclusive group entrance, the algorithm must ensure that each node properly completes its participation in the group, receiving an aggregation result that has taken into account its estimated value. Recall that in this model messages are reliably transmitted in FIFO order but can suffer arbitrary finite delays.

Since leaders initiating a GCM call cannot anticipate how many nodes will acknowledge them, there is still a need to consider some timeout for the reception of the JACK responses. This opens the possibility that some nodes that have sent a JACK message will not have this message, and mass, processed in this iteration of the protocol. It is thus necessary that GAM messages are augmented with node and iteration ids, so that only nodes whose JACK reached the leader will consider the subsequent GAM. Iteration ids will also be important so that leaders discard JACK messages from previous GCMs that they initiated.

One can conclude that although there seems to exist no important obstacles for an implementations under this model, several modifications should be considered and the correctness of the resulting approach carefully examined.

Message Loss In terms of message loss, the authors consider the occurrence of collisions and link failures, but they only consider its effect on GMCs, with impacts on the creation of groups, reducing its expected size. For instance, they assume that link failures only happen between iterations of the protocol, which is unrealistic since links can unpredictably fail at any point of the algorithm execution, provoking the loss of any type of message.

In this setting, timeouts are needed in the receptions steps for expected JACK and GAM messages. The loss of a GCM will have no impact on the correctness of the algorithm, only preventing nodes from joining the group. However, loosing a JACK from a node but delivering the subsequent GAM to that same node will most probably violate mass conservation. The same happens if GAMs are lost.

Extensions that assign iteration ids and node ids to the protocol messages can possibly address some of these issues, but the need to have a successful GAM reception on nodes that contributed their mass in a successfully delivered JACK message points again to the *coordinated attack problem*.

Node Crash Similarly to previous approaches, this algorithm does not support node crashes (not addressed by the authors). The impact of the removal of a node from the network will be proportional to the contribution of its value to the computation of the global aggregate. As already mentioned before, the unpredicted failure of a node will alter the total mass of the system, breaking the mass conservation invariant, and leading the system to converge to a wrong result.

4 Case Study: Push-Pull Gossiping

In the previous section, we argue that most aggregation algorithms by averaging have practical implementation issues in realistic environments. In order to solve the exposed issues and ensure the algorithms dependability, important modifications must be introduced to their implementation. However, these additional modifications will incur in some degradation of their performance. We now choose to address in more detail PPG, since it is the only one that does not converge to the true aggregation result in a synchronous operation mode, as depicted by simulation results of Figure 2.

We use a custom high level simulator to evaluate the execution of the analyzed algorithms in a synchronous operation mode[2], providing a fair comparison of them. However, the simulation level was detailed enough to observe some effects on messages passing (in particular, interleaving), since we assume that all the messages received in a common round will be processed by the target node

[2] synchronous model as in Chapter 2 of [17]

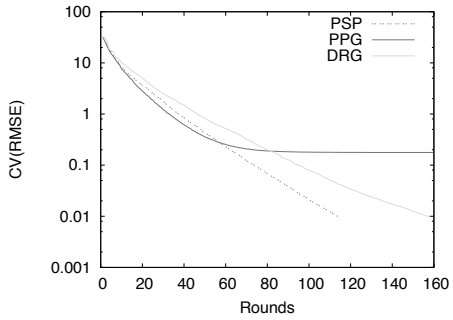


Fig. 2. Convergence speed of the analyzed algorithms (PSP, PPG and DRG).

in some arbitrary order. The depicted results correspond to the average values (number of rounds or messages) required to reach a specific accuracy (coefficient of variation of the root mean square error), obtained from 50 trials of the execution of the algorithms under identical settings. In each trial different networks with a random topology (generated according to the Erdős–Rényi model [20]) and the same characteristics (network of size 1000 and average connection degree of 5) are used. We implement the COUNT version of each simulated algorithm (to determine the network size) and tune all its parameters (e.g. the probability to become leader in DRG) to obtain its best performance in each simulation scenario.

As previously referred (in Section 3.2) and confirmed by simulations (see Figure 2), the correctness of PPG (mass conservation) is affected by the occurrence of messages interleaving in push-pull operations, occurring even in synchronous rounds settings. In practice, in order to guaranty the convergence of PPG to the correct result, an additional property must be considered: the push-pull process must be atomic, and the approximation value cannot be update by any concurrent process before the end of an already initiated push-pull procedure. We propose two concrete alternative versions of the algorithm that tackle this problem, and guarantee the consistency of the push-pull process: *Push-Pull Back Cancellation* (PPBC), and *Push-Pull Ordered Wait* (PPOW).

4.1 Push-Pull Back Cancellation

This version of the algorithm is based on a message cancellation mechanism, to guaranty that no other process will interfere in an ongoing push-pull operation. In this scheme, after sending a *push* message to the chosen target, all the received messages will be ignored (without any local state update) by the source node, until the awaited *pull* message is received. The cancelation mechanism is implemented by sending a *pull* message with the same value received in the *push* one, skipping the correspondent state update. By reflecting back the received approximation value, the update performed by the initiator will not change its approximation value. The main drawback of this mechanism is the execution of

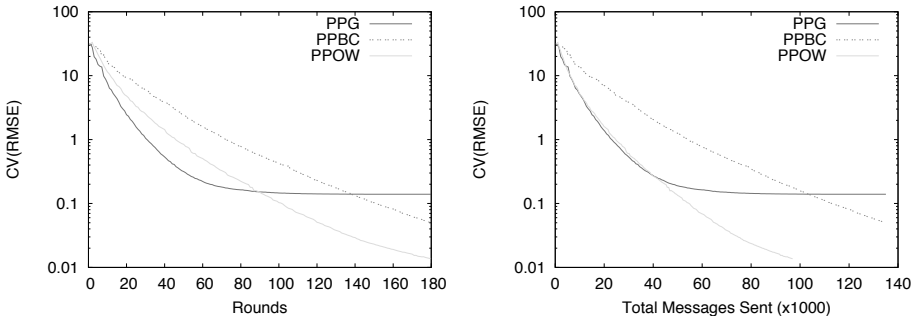


Fig. 3. New proposed version of PPG compared to the original algorithm.

dummy push-pulls (that will not contribute to the system convergence), all in order to ensure a consistent state update of ongoing push-pull operations.

4.2 Push-Pull Ordered Wait

This version of the algorithm adds a message buffering process to the original version of the algorithm. All *push* messages received while a push-pull operation has already been initiated by a node are locally buffered, and will only be processed after the conclusion of the ongoing push-pull. Notice that, simply considering this buffering mechanism could incur in deadlocks across the network, due to possible creation of waiting cycles between nodes. E.g. If node n_1 is waiting for node n_2, which is waiting for node n_3, and by is turn n_3 is waiting for n_1, ending all locked waiting for each other.

Additional requirements must be considered to avoid this cyclic waiting situation. For instance, we define a total order between nodes (by setting a UID to each node), and stipulate that each node can only initiate a push-pull operation with nodes from a specific order (inferior or superior). For example, considering the order between nodes $n_1 < n_2 < n_3$ (defined by their UID, from lower to higher ID values), and imposing that *push* messages can only be sent to nodes with a higher ID: node n_3 could never be waiting for n_1 (breaking the previous cycle), since it has a lower ID and consequently no *push* messages can be sent to it by n_3. Although, this scheme restrains the execution of push-pull in some directions, unlike the previous version it does not wast concurrent push-pulls, and makes the most of their contributions to the convergence of the algorithm.

4.3 Simulation Results

As depicted by Figure 3, the modifications introduced to the original algorithm solve the convergence problem of the algorithm, but at the cost of a performance degradation in terms of convergence speed (see Figure 3(a)). As expected PPOW

performs better than PPBC, since its buffering mechanism allows the integration of all the received contributions to the aggregation process. By making all contributions (messages) count, PPOW exhibits similar results than the original algorithms in terms of overhead (number of message needed to reach a common accuracy), as showed in Figure 3(b).

5 Conclusion

In this study, we expose important implementation and dependability issues in averaging based aggregation algorithms. Issues that are often overlooked in the abstract modeling of the algorithms and that must be addressed in any concrete real scenario.

In particular, we discuss three established algorithms from this class under practical settings, namely the *Push-Sum Protocol* [1], *Push-Pull Gossip protocol* [2] and *Distributed Random Grouping* [3]. All algorithms evidence some dependability issues, that compromise their correctness, when exposed to asynchronous and faulty (e.g. with message loss or node crash) environments.

Supplementary mechanisms and modifications must be added to the discussed aggregation algorithms, in order to provide fault tolerance and enable their practical use. These additional provisions will result in a degradation of the algorithms performance. In particular, we extend the *Push-Pull Gossip protocol*, which exhibit issues even on synchronous settings, and propose two concrete version of the protocol to solve its interleaving problems: *Push-Pull Back Cancellation*, and *Push-Pull Ordered Wait*. As showed by the results obtained from simulations, the proposed algorithms solve the convergence problem of the original algorithm, but exhibit a worse global performance (especially in terms of convergence speed).

The depicted vulnerability of current averaging algorithms, and their importance when seeking high precision aggregates, also served as a motivation for our ongoing research on *Flow Updating* [21], a fast fault tolerant averaging algorithm that is, by design, resilient to message loss.

References

1. D Kempe, A Dobra, and J Gehrke. Gossip-based computation of aggregate information. *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 482–491, 2003.
2. M Jelasity, A Montresor, and O Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3):219 – 252, Aug 2005.
3. Jen-Yeu Chen, G Pandurangan, and Dongyan Xu;. Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis. *IEEE Transactions on Parallel and Distributed Systems*, 17(9):987 – 1000, 2006.
4. Robbert Van Renesse. The importance of aggregation. *Future Directions in Distributed Computing, Lecture Notes in Computer Science*, 2584:87–92, 2003.

5. M Jelasity and A Montresor. Epidemic-style proactive aggregation in large overlay networks. *24th International Conference on Distributed Computing Systems*, pages 102 – 109, Jan 2004.
6. Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, Dec 2002.
7. S Madden, R Szewczyk, M Franklin, and D Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications*, pages 49–58, 2002.
8. J Li, K Sollins, and D Lim. Implementing aggregation and broadcast over distributed hash tables. *ACM SIGCOMM Computer Communication Review*, 35(1):81–92, Jan 2005.
9. Yitzhak Birk, Idit Keidar, Liran Liss, Assaf Schuster, and Ran Wolff. Veracity radius: capturing the locality of distributed computations. *PODC '06: Proceedings of the 25th annual ACM symposium on Principles of distributed computing*, 2006.
10. Fetahi Wuhib, Mads Dam, Rolf Stadler, and Alexander Clemm. Robust monitoring of network-wide aggregates through gossiping. *10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 226 – 235, 2007.
11. Carlos Baquero, Paulo Sérgio Almeida, and Raquel Menezes. Fast estimation of aggregates in unstructured networks. *International Conference on Autonomic and Autonomous Systems (ICAS), Valencia, Spain*, Apr 2009.
12. D Mosk-Aoyama and D Shah. Computing separable functions via gossip. *PODC '06: Proceedings of the 25th annual ACM symposium on Principles of Distributed Computing*, pages 113–122, 2006.
13. Laurent Massoulié, Erwan Merrer, Anne-Marie Kermarrec, and Ayalvadi Ganesh. Peer counting and sampling in overlay networks: random walk methods. *PODC 06: Proceedings of the 25th annual ACM symposium on Principles of Distributed Computing*, 2006.
14. A Ganesh, A Kermarrec, E Le Merrer, and L Massoulié. Peer counting and sampling in overlay networks based on random walks. *Distributed Computing*, 20(4):267–278, Nov 2007.
15. Sandeep Mane, Sandeep Mopuru, Kriti Mehra, and Jaideep Srivastava. Network size estimation in a peer-to-peer network. *Technical report, Department of Computer Science - University of Minnesota*, page 12, Sep 2005.
16. D Kostoulas, D Psaltoulis, Indranil Gupta, K Birman, and Al Demers. Decentralized schemes for size estimation in large and dynamic groups. *4th IEEE International Symposium on Network Computing and Applications*, pages 41–48, 2005.
17. Nancy A Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.
18. Lorenzo Alvisi, Jeroen Doumen, Rachid Guerraoui, Boris Koldehofe, Harry Li, Robbert Renesse, and Gilles Tredan. How robust are gossip-based communication protocols? *ACM SIGOPS Operating Systems Review*, 41(5), Oct 2007.
19. Jim Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course. Lecture Notes in Computer Science, Vol. 60*, pages 393–481, London, UK, 1978. Springer-Verlag.
20. Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.
21. Paulo Jesus, Carlos Baquero, and Paulo Sérgio Almeida. Fault-tolerant aggregation by flow updating. In *9th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*, volume 5523 of *Lecture Notes in Computer Science*, pages 73–86, Lisbon, Portugal, June 2009. Springer.

D^{2}STM: Memória Transacional em Software Distribuída e Confiável*

Maria Couceiro, Paolo Romano, Nuno Carvalho, and Luis Rodrigues

INESC-ID/IST

maria.couceiro@ist.utl.pt, romanop@gsd.inesc-id.pt,
nonius@gsd.inesc-id.pt, ler@ist.utl.pt

Resumo Os sistemas de Memória Transacional em Software (MTS) têm vindo a ganhar relevo no desenvolvimento de aplicações concorrentes. No entanto, o recurso à distribuição e à replicação neste contexto não se encontra ainda devidamente explorado. Este artigo apresenta o D^{2}STM, um sistema de MTS replicada que rentabiliza os recursos disponíveis nos vários nós de um sistema distribuído. A coerência da MTS replicada é assegurada de modo transparente, mesmo na presença de faltas no sistema. No D^{2}STM, as transacções são processadas de forma autónoma por um único nó, evitando-se qualquer comunicação entre réplicas durante a sua execução. Um processo distribuído de certificação não-bloqueante, ao qual se deu o nome de BFC (*Bloom Filter Certification*), assegura a coerência no momento de *confirmação* da transacção. O BFC explora os mecanismos de codificação dos filtros de Bloom para reduzir substancialmente o custo inerente à coordenação das várias réplicas (à custa de um aumento da probabilidade de uma transacção ser abortada, com um valor que pode ser ajustado pelo utilizador). Recorrendo a uma bancada experimental mostra-se que o BFC permite atingir ganhos consideráveis.

Abstract. Software Transactional Memory (STM) systems have emerged as a powerful paradigm to develop concurrent applications. This paper addresses the problem of building a distributed and replicated STM system. We present D^{2}STM, a replicated STM, where consistency is ensured in a transparent manner, even in the presence of failures. In D^{2}STM transactions are autonomously processed on a single node, avoiding any replica inter-communication during transaction execution. Consistency is enforced at transaction commit time by a non-blocking distributed certification scheme, which we name BFC (Bloom Filter Certification). BFC exploits a novel Bloom Filter-based encoding mechanism that permits to significantly reduce the overhead of replica coordination (at the cost of a user tunable increase in the probability of transaction abort). Experimental results based on STM benchmarks show that the BFC scheme permits to achieve remarkable performance gains.

* Este trabalho foi parcialmente suportado pelo projecto Pastramy (PTDC/EIA/72405/2006).

1 Introdução

Os sistemas de Memória Transacional em Software (MTS) têm vindo a ganhar relevância no desenvolvimento de aplicações concorrentes [1,2]. Os programadores que utilizam MTSs não necessitam de lidar explicitamente com mecanismos de controlo de concorrência, tendo apenas de identificar a sequência de instruções, ou transacções, que acedem ou modificam objectos partilhados de forma concorrente e que necessitam de ser executadas de forma atómica. Aumenta-se, assim, a fiabilidade do código, ao mesmo tempo que se diminui o tempo de desenvolvimento.

Apesar do interesse que as MTSs têm suscitado, só muito recentemente se começou a abordar o problema da construção de arquitecturas distribuídas para este tipo de sistemas [3,4,5]. Para além disso, as soluções anteriormente propostas recorrem à replicação como forma de melhorar o desempenho e não exploram o facto de esta poder também ser usada para obter tolerância a faltas. No entanto, este é um aspecto central no desenho das MTS distribuídas, uma vez que a probabilidade de ocorrência de falhas aumenta com o número de nós, sendo incontornável num sistema de grande dimensão. As garantias de coerência forte e tolerância a faltas são também essenciais quando as MTSs são usadas para aumentar a robustez das aplicações orientadas a serviços. Como exemplo, cita-se o sistema FenixEDU [6], uma aplicação web na qual a semântica transacional das operações é suportada por uma MTS.

Neste trabalho apresenta-se o D^{2}STM, um sistema de MTS tolerante a faltas e distribuído que permite tirar partido dos recursos computacionais disponíveis num *cluster*, usando a interface convencional de uma MTS e assegurando de modo transparente a coerência dos dados de forma não bloqueante, mesmo em caso de falhas.

O esquema de sincronização de réplicas usado no D^{2}STM é inspirado por trabalhos recentes na área da replicação de bases de dados [7,8,9], os quais recorrem às propriedades da primitiva de difusão atómica [10] para manter a coerência entre réplicas através de um processo de certificação distribuído. Este tipo de certificação permite que as transacções sejam executadas localmente de forma optimista, sendo a coerência das réplicas (normalmente *1-Copy serializability*) assegurada no momento da confirmação da transacção. Esta fase de certificação distribuída recorre às propriedades da primitiva de difusão atómica para que as transacções tenham uma ordem de seriação comum a todas as réplicas. Para além disso, apenas uma réplica executa toda a transacção (de escrita), enquanto que as restantes têm somente de a validar e guardar as actualizações resultantes. Esta abordagem permite uma capacidade de escala elevada, mesmo na presença de padrões de utilização dominados por operações de escrita, desde que a taxa de conflitos entre transacções não atinja valores muito elevados [7].

Apesar do uso deste tipo de certificação parecer o mais indicado para aplicar em MTSs, resultados anteriores [11] (confirmados pelos resultados experimentais apresentados neste trabalho) mostram que o uso de primitivas de difusão atómica pode ser extremamente penalizante neste contexto. Uma vez que, num sistema baseado em MTSs, não existem necessariamente acessos ao disco, nem

o processamento relacionados com a análise de comandos SQL ou com a optimização de planos de execução, o tempo de execução de uma transacção numa MTS é significativamente mais curto do que num sistema de base de dados clássico. Com o objectivo de minimizar o impacto da coordenação inter-réplicas, o D^{2}STM introduz um novo processo de certificação, com o nome de BFC (*Bloom Filter Certification*). O BFC tira partido das técnicas de codificação dos filtros de Bloom para reduzir de forma significativa quer o processamento associado à certificação distribuída, quer o tamanho das mensagens trocadas através da primitiva de difusão atómica. Estas melhorias são atingidas com o sacrifício de um aumento marginal (mas configurável pelo utilizador) da probabilidade de uma transacção abortar.

O D^{2}STM foi construído sobre a JVSTM [12], uma biblioteca MTS que suporta controlo de concorrência multi-versão e, como consequência, oferece um excelente desempenho para transacções em que só se executam leituras, uma vez que estas se encontram protegidas contra a possibilidade de abortarem devido a conflitos (locais ou remotos). Através de uma extensa avaliação experimental, baseada não só em testes localizados mas também em bancadas de teste mais complexas, mostra-se que o D^{2}STM permite atingir ganhos consideráveis no desempenho, sem aumentar de forma significativa a probabilidade de uma transacção abortar.

O resto do artigo está organizado da seguinte forma. Na Secção 2 discute-se o trabalho relacionado. Na Secção 3 apresenta-se o modelo do sistema e na Secção 4 mostra-se uma visão geral da arquitectura do sistema, bem como a integração da JVSTM no D^{2}STM. O BFC é apresentado na Secção 5 e na Secção 6 mostram-se os resultados da avaliação experimental. Finalmente, a Secção 7 conclui o artigo.

2 Trabalho Relacionado

Nesta secção faz-se uma síntese do trabalho relevante para a construção do D^{2}STM. Começa-se por analisar sistemas de MTS distribuída, tendo em conta alguns aspectos fundamentais, como a tolerância a faltas e desempenho. Apresenta-se também uma discussão das vantagens e desvantagens de recentes algoritmos de replicação de bases de dados quando aplicados no contexto de MTSs distribuídas.

2.1 MTSs Distribuídas

As únicas soluções de MTSs distribuídas das quais temos conhecimento estão descritas em [3,4,5] e, como já foi mencionado na introdução, nenhuma delas tira partido da replicação como meio de assegurar coerência e disponibilidade quando ocorrem faltas. Por oposição, a tolerância a faltas foi tida em conta durante o desenho do D^{2}STM, estando estes mecanismos (como, por exemplo, a difusão atómica) integrados com um novo e eficiente processo distribuído de certificação de transacções. Segue-se uma descrição das principais diferenças entre os processos de manutenção de coerência entre réplicas adoptados por estas soluções.

O trabalho apresentado em [3] utiliza um algoritmo multi-versão distribuído (DMV, *Distributed Multi Versioning*) e explora a presença simultânea de várias versões do mesmo conjunto de dados nas diferentes réplicas. O DMV permite que transacções de leitura sejam executadas em paralelo com as de escrita (podendo ocorrer conflitos), à semelhança de outros algoritmos de controlo de concorrência multi-versão centralizados [13] (onde se inclui a JVSTM [12]). No entanto, cada réplica mantém uma única versão dos dados e atrasa a sua actualização para diminuir a probabilidade de abortar transacções de leitura em execução. O D^{2}STM tem como base uma MTS multi-versão (JVSTM) que, para cada objecto transacional, mantém um número de versões suficiente para garantir que nenhuma transacção de leitura é abortada. Adicionalmente, o DMV requer que cada transacção que pretende confirmar adquira previamente um testemunho (*token*) único no sistema para obter uma seriação global das transacções que confirmam, introduzindo um atraso considerável [4]. Por oposição, no D^{2}STM as réplicas têm apenas de executar uma validação local da transacção após a fase de coordenação, que tem como base uma primitiva de difusão atómica e pode ser executada de forma concorrente por todas as réplicas.

O trabalho descrito em [4] não recorre a algoritmos multi-versão mas, analogamente a [3], utiliza um mecanismo distribuído de exclusão mútua. Estes mecanismos têm como objectivo assegurar que duas réplicas nunca irão confirmar simultaneamente transacções com conflitos entre si. O uso de múltiplas *leases* com base no conjunto de dados acedidos pelas transacções permite atenuar os problemas de desempenho causados pela seriação da fase de confirmação distribuída. No entanto, a sua atribuição é coordenada por uma única réplica, o que diminui a capacidade de escala da solução (comprovado pelo facto de a avaliação experimental em [4] não apresentar resultados para mais de quatro réplicas).

Finalmente, o Cluster-STM, apresentado em [5], centra-se no problema de particionar o conjunto de dados pelos nós de um MTS distribuído de grande escala. Cada item de dados transaccional tem atribuído um nó responsável não só por manter a cópia principal dos dados (isto é, aquela que corresponde sempre à versão mais actual dos dados), mas também por sincronizar os acessos de transacções remotas que possam gerar conflitos. No entanto, os esquemas de replicação são delegados para o nível da aplicação, aumentando assim a sua complexidade. Para além disso, processadores no mesmo nó ou em nós diferentes são tratados da mesma forma, não sendo explorada a partilha de memória entre múltiplos processadores para tornar mais rápida a comunicação dentro do mesmo nó.

2.2 Replicação em Bases de Dados

A maioria dos algoritmos de replicação de bases de dados recentes [7,8,9] recorrem a uma primitiva de difusão atómica [10,14], tipicamente disponibilizada por um Serviço de Comunicação em Grupo (SCG) [15,16]. Esta primitiva assegura uma ordem global de seriação de transacções, sem os problemas de interbloqueio e falta de capacidade de escala que caracterizam os mecanismos clássicos de replicação [17].

As abordagens optimistas, como por exemplo [7], recorrem à execução das transacções numa única réplica e à sua posterior validação através de um processo de certificação global (baseado na difusão atómica) que permite detectar conflitos entre transacções concorrentes. Estas abordagens podem ser divididas em dois grandes grupos [8], nomeadamente (*i*) nas que não incluem um processo de votação (envia-se tanto o conjunto de escritas como o de leituras de cada transacção) e (*ii*) nas que o incluem (onde só se envia o conjunto de escritas da transacção mas é necessário transmitir uma segunda mensagem durante a fase de confirmação [14]). Dado que o tempo de execução de uma transacção é, em média, mais curto numa MTS do que num sistema de base de dados, o atraso induzido pela coordenação de réplicas é relativamente elevado [11]. Isto implica que os processos de certificação que incluem votação não sejam os mais indicados para aplicar neste contexto, uma vez que introduzem um passo extra na comunicação. Por outro lado, iremos mostrar na avaliação experimental que a eficiência dos protocolos sem votação é profundamente afectada pelo tamanho do conjunto de leituras das transacções.

O esquema de coordenação de réplicas utilizado pelo D^{2}STM, ao qual se deu o nome de BFC (*Bloom Filter Certification*), pode ser classificado como um algoritmo de certificação sem votação. O conjunto de leituras das transacções é comprimido num filtro de Bloom de modo a ser possível fazer apenas uma difusão atómica e não congestionar a rede com mensagens demasiado extensas. Isto é conseguido à custa de um ligeiro aumento na probabilidade de uma transacção ser abortada (definido pelo utilizador).

3 Modelo do Sistema

Consideramos um sistema distribuído assíncrono clássico com $\Pi = \{p_1, \ldots, p_n\}$ processos que comunicam através de mensagens e podem falhar de acordo com o modelo de falha por paragem. Assumimos que a maioria dos processos está correcta e que o sistema assegura um nível de sincronismo suficiente (por exemplo, através de um detector de falhas $\Diamond S$) que permite concretizar um serviço de difusão atómica com as propriedades de validade, acordo, integridade e ordem total uniforme [10].

O critério de consistência para o estado das instâncias MTSs replicadas assegurado pela D^{2}STM corresponde a *1-copy serializability* de leituras e escritas de dados transaccionais [13], ou seja, o histórico da execução de um conjunto de transacções num conjunto de réplicas é equivalente ao histórico da sua execução em série numa única instância de MTS não replicada.

4 Arquitectura do D^{2}STM

4.1 Componentes

Como a Figura 1 ilustra, um nó do sistema D^{2}STM encontra-se estruturado em quatro camadas lógicas. A camada de baixo é materializada por um serviço de

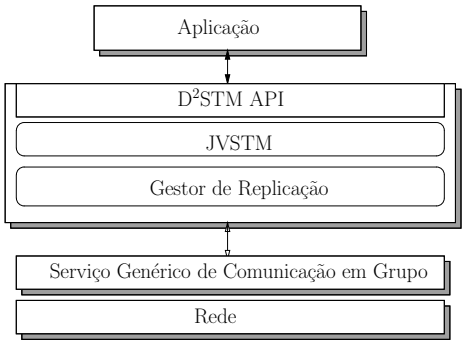


Figura 1. Componentes de uma réplica D²STM.

comunicação de grupo (SCG) [10] que fornece uma primitiva de difusão atómica, entre outras. A nossa concretização usa um SCG genérico [18], que suporta várias concretizações do SCG (todos os testes descritos neste artigo foram realizados com o SCG Appia [15]). O componente principal do D²STM é representado pelo Gestor de Replicação que concretiza o protocolo de coordenação distribuído necessário para assegurar a coerência das réplicas (isto é, *1-copy serializability*); na Secção 5 encontra-se uma descrição detalhada deste componente. Por fim, a camada superior da D²STM é um contentor que intercepta chamadas a marcadores de transacção (isto é, indicações para iniciar e terminar transacções) do nível da aplicação sem interferir com os acessos da aplicação (leitura e escrita) aos objectos que são geridos directamente pela camada inferior (JVSTM). Esta abordagem permite à D²STM estender o modelo clássico de programação de MTSs de forma transparente, não sendo necessário realizar modificações significativas às aplicações JVSTM pré-existentes.

4.2 Integração com a JVSTM

A JVSTM usa um algoritmo multi-versão que se baseia na abstracção de *versioned box* (VBox) para guardar o estado mutável de um programa concorrente. Uma VBox é um contentor que guarda uma sequência de valores etiquetados, isto é, o seu histórico. Cada um destes valores corresponde a uma alteração à *box* feita por uma transacção confirmada e é etiquetada com o número de versão (inteiro incrementado a cada confirmação) da transacção correspondente. Cada transacção é iniciada com o número de versão da última transacção confirmada com sucesso (*commitTimestamp*). Esta informação é usada durante a execução da transacção (para identificar os valores a ler das VBoxes) e também quando esta se prepara para confirmar, durante a fase de validação (para determinar o conjunto de transacções concorrentes e verificar se ocorreu algum conflito).

A abordagem optimista da JVSTM consiste em diferir as actualizações de uma transacção para fazer a detecção de conflitos apenas durante a confirmação, verificando se algumas das *VBoxes* lidas por uma transacção T foram modificadas por outra transacção T' com um número de versão superior. Em caso

afirmativo, T é abortada. Caso contrário, o número de versão de T é actualizado com o valor de *commitTimestamp* e os novos valores das VBoxes que actualizou são atomicamente guardados nas respectivas VBoxes.

De modo a aumentar o desempenho, o protocolo de coordenação de réplicas do D^{2}STM, ou seja, o BFC, está intrinsecamente integrado no mecanismo de gestão de versões da JVSTM. Os pormenores desta integração podem ser consultados em [19].

5 Certificação com Filtros de Bloom

A certificação com filtros de Bloom (BFC) é um novo processo de certificação sem votação que explora as propriedades destes filtros [20], de modo a reduzir drasticamente o custo da fase de certificação distribuída com o sacrifício de um aumento reduzido (e controlado) no risco de uma transacção abortar.

Um filtro de Bloom que representa um conjunto $S = \{x_1, x_2, \ldots, x_n\}$ com n elementos consiste num vector de m dígitos binários inicializados a 0 [21] . O filtro usa k funções de dispersão $h_1, \ldots, h_k$ no intervalo $\{1, \ldots, m\}$. Se o conjunto não estiver vazio, para cada elemento $x \in S$, os dígitos $h_i(x)$ são colocados a 1 para $1 \leq i \leq k$. Para verificar se um item y pertence a S, todos as posições $h_i(y)$ têm de estar a 1. Se isto não se verificar, então y não pertence a S. Caso contrário, assume-se que y pertence a S, com uma probabilidade de erro conhecida. A taxa de falsos positivos para uma única interrogação a um filtro depende do número de dígitos usados por item m/n e do número de funções de dispersão k, de acordo com a seguinte equação:

$$f = (1 - e^{-kn/m})^k \tag{1}$$

onde o número de funções de dispersão k óptimo que minimiza a probabilidade de falsos positivos dado por m e n é igual a:

$$k = \lceil ln\ 2 \cdot m/n \rceil \tag{2}$$

No BFC, as transacções de leitura são executadas localmente e a confirmação é feita sem nenhum custo adicional, tirando-se partido do algoritmo multi-versão da JVSTM para assegurar que estas transacções nunca são abortadas (devido a conflitos locais ou remotos).

Após a sua execução, uma transacção com um conjunto de escritas não nulo (ou seja, que actualizou pelo menos uma VBox) passa por uma fase de validação local (de modo a evitar a fase distribuída do processo de certificação em transacções quando é possível determinar localmente o resultado). Se a transacção passa esta fase, o Gestor de Replicação codifica o conjunto de leituras da transacção (ou seja, o conjunto de identificadores das *VBoxes* lidas durante a sua execução) num filtro de Bloom e, juntamente com o conjunto de escritas da transacção (este não codificado no filtro), envia-o através de difusão atómica.

Tal como nos protocolos de certificação sem votação clássicos, as transacções de escrita são validadas aquando da sua entrega (por difusão atómica). Verifica-se, então, se o filtro de Bloom de uma transacção T_x contém algum item ac-

tualizado por transacções com um número de versão maior que T_x. Se isto se verificar, então T_x é abortada. Caso contrário, T_x pode ser confirmada.

Uma vez que a fase de validação de uma transacção T_x necessita dos conjuntos de escritas das transacções concorrentes já confirmadas, o Gestor de Replicação guarda os identificadores das *VBoxes* actualizadas por elas. De modo a evitar manter em memória informação sobre transacções que já não afectam nenhuma transacção activa, recorremos a um processo de reciclagem de memória distribuído (análogo ao usado em [22]) no qual cada réplica dá a conhecer o menor número de versão das suas transacções activas (juntando-o às mensagens de validação das transacções), sendo assim possível qualquer réplica determinar qual a transacção mais antiga em execução no sistema.

Para uma transacção T_x ser abortada devido a um falso positivo, basta que este ocorra em qualquer um dos itens actualizados por transacções concorrentes. Logo, para determinar o tamanho do filtro de uma transacção seria necessário saber exactamente o número de interrogações q que lhe iriam ser feitas durante a fase de validação, de modo a que nunca se exceda a taxa de aborto máxima definida pelo utilizador (*maxAbortRate*). Por outro lado, no momento em que T_x inicia a fase de validação, é impossível prever o número e o tamanho dos conjuntos de escritas de transacções que entretanto confirmaram. No entanto, erros que possam ocorrer na estimativa de q resultam apenas em desvios (positivos ou negativos) na *maxAbortRate*. Por isso, q é estimado através da média do número de interrogações feitas a um filtro durante a fase de validação das últimas t transacções (em que t é um valor pré-definido), já que este número é facilmente obtido por todas as réplicas. A partir de q, podemos determinar o número de dígitos m, considerando que os falsos positivos para cada interrogação distinta são eventos independentes e identicamente distribuídos que geram um processo de Bernoulli [23]. Assim, a probabilidade de uma transacção abortar devido a um falso positivo no processo de validação baseado em filtros de Bloom pode ser representada por:

$$maxAbortRate = 1 - (1 - f)^q$$

que, combinada com as equações 1 e 2, nos permite estimar m como:

$$m = \left\lceil - n \, \frac{log_2(1 - (1 - maxAbortRate)^{\frac{1}{q}})}{ln\ 2} \right\rceil$$

Podemos provar (informalmente) a correcção do BFC tendo em consideração que (*i*) as réplicas validam todas as transacções de escrita pela mesma ordem (determinada pela primitiva de difusão atómica) e que (*ii*) o processo de validação é determinista, uma vez que todas elas utilizam o mesmo conjunto de funções de dispersão na construção/descodificação dos filtros de Bloom. Assim, a ocorrência de falsos positivos durante o processo de validação apenas resulta no aumento da probabilidade de uma transacção abortar e nunca em incoerências no estado das réplicas.

6 Avaliação

Apresentamos agora os resultados da avaliação experimental realizada. Esta pretende determinar os ganhos no desempenho quando o BFC é aplicado num sistema de MTS distribuído real, ou seja, usando um protótipo do D^{2}STM com cargas sintéticas e com padrões de carga complexos. Os testes foram realizados num sistema com 8 nós, cada um deles equipado com um Intel QuadCore Q6600 a 2.40GHz com 8 GB de RAM e Linux 2.6.27.7 ligado via Ethernet Gigabit privada. A concretização do protocolo de difusão atómica usada foi baseada no algoritmo clássico que usa um sequenciador [14,10].

Começa-se por considerar uma carga sintética (adaptada do *Bank Benchmark* originalmente usada para avaliar a DSTM2 [24]). Cada réplica é iniciada com um vector de $numThreads \cdot numMachines \cdot 10.000$ itens e cada fio de execução acede a um fragmento distinto de 10.000 elementos do vector, lendo-os a todos e aleatoriamente actualizando um número de elementos uniformemente distribuído no intervalo [50, 100]. Como diferentes fios de execução acedem a fragmentos do vector que não se sobrepõem, os falsos positivos da validação com filtros de Bloom são a única razão pela qual as transacções são abortadas. O gráfico da Figura 2 mostra a percentagem de transacções abortadas quando se usa o BFC com *maxAbortRate* de 1%, 5%, 10%, variando-se o número de réplicas entre 1 e 8 (com 4 fios de execução em cada réplica), e ilustra a correspondência entre a previsão analítica e os resultados experimentais.

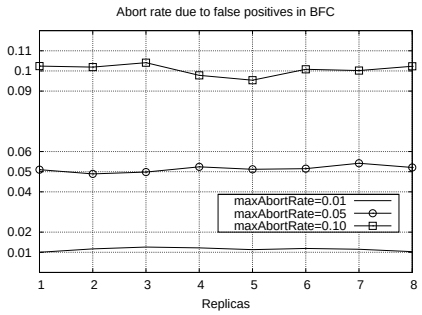
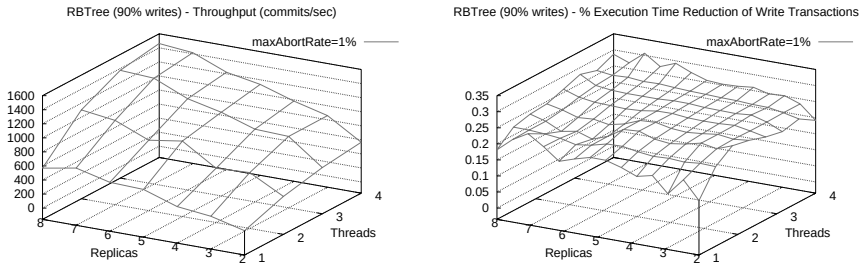


Figura 2. Taxa de cancelamento de transacções devido a falsos positivos.

Apresentam-se agora resultados para uma *micro-bancada* de teste mais complexa, denominada *Red Black Tree* (novamente adaptada de uma implementação usada para avaliar a DSTM2 [24]). Existem três tipos diferentes de transacções: i) leitura que executa uma sequência de pesquisas, ii) escrita que executa uma sequência de pesquisas e inserções, e iii) escrita que executa uma sequência de pesquisas e remoções. A bancada foi configurada (os detalhes de configuração usados nestes e noutros testes podem ser consultados em [19]) de modo a que a probabilidade de contenção de leitura-escrita seja baixa e com um padrão de

utilização de 90% de transacções de escrita. A Figura 3(a) mostra o débito do sistema variando o número de réplicas entre 2 e 8 e o número de fios de execução entre 1 e 4. É possível observar uma melhoria linear no desempenho com o aumento do número de réplicas, chegando mesmo a duplicar entre a utilização de 2 e 6 réplicas. A Figura 3(b) mostra a redução do tempo de execução de transacções de escrita com o BFC quando comparado com um processo de certificação sem votação clássico, em tudo semelhante ao BFC, mas no qual se enviam os conjuntos de leitura e escrita sem qualquer tipo de compressão. Pode-se observar que os ganhos obtidos com o BFC variam entre os 20% e os 30%.

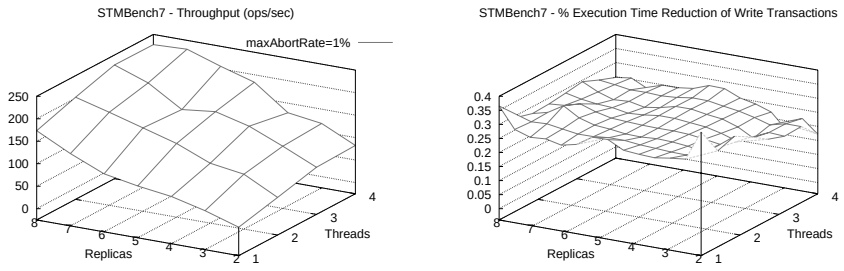


(a) Débito (b) Redução no tempo de execução (%)

Figura 3. Red Black Tree, 90% de escritas, $maxAbortRate$=1%

Apresentam-se finalmente resultados obtidos usando a STMBench7 [25]. A Figura 4 mostra o desempenho do sistema quando usada a carga denominada *"read dominated with long traversals"*. O gráfico mostra o débito para um número de réplicas entre 2 e 8, cada uma com 1 a 4 fios de execução. Os resultados observados no aumento do débito são coerentes com os obtidos na bancada de testes anterior. A Figura 4(b) mostra os ganhos no desempenho obtidos com o BFC quando comparado com o processo de certificação sem votação clássico já referido. É possível observar que o tempo de execução das transacções de escrita (ou seja, as que passam pelo processo de certificação distribuído) apresenta reduções que variam entre 20% e 40%. Um facto interessante sublinhado por estes testes é o facto de o BFC atingir ganhos no desempenho consideráveis mesmo com um aumento na probabilidade de uma transacção abortar muito pequeno (1%).

Podemos assim concluir que o BFC concretizado na D^{2}STM assegura tolerância a faltas, torna possível o uso de réplicas adicionais para melhorar o débito do sistema (principalmente na presença de cargas em que predominem operações de leitura) e, por fim, permite o uso de um processo de certificação sem votação mais rápido na presença de cargas com conjuntos de leitura de grandes dimensões.



(a) Débito (b) Redução no tempo de execução (%)

Figura 4. STMBench7, *read dominated with long traversals*, $maxAbortRate$=1%

7 Conclusões

Este trabalho apresentou o D^{2}STM, um sistema de memória transaccional em software distribuída que assegura coerência forte e alta disponibilidade mesmo na presença de falhas em (uma minoria de) réplicas. O BFC, algoritmo que mantém a coerência entre as réplicas utilizado pelo D^{2}STM, tira partido dos filtros de Bloom para reduzir de modo significativo o custo associado à fase de certificação das transacções. Para além disso, graças à integração com uma MTS multi-versão, o D^{2}STM processa as transacções de leitura localmente sem o risco destas serem abortadas devido a algum conflito local ou remoto, evitando-se ao mesmo tempo a comunicação desnecessária entre réplicas.

Referências

1. Herlihy, M., Luchangco, V., Moir, M., Scherer, III, W.N.: Software transactional memory for dynamic-sized data structures. In: Proc. of the Symposium on Principles of Distributed Computing (PODC), ACM (2003) 92–101
2. Fraser, K.: Practical lock freedom. PhD thesis, Cambridge University Computer Laboratory (2003) Also available as Technical Report UCAM-CL-TR-579.
3. Manassiev, K., Mihailescu, M., Amza, C.: Exploiting distributed version concurrency in a transactional memory cluster. In: Proc. of the Symposium on Principles and Practice of Parallel Programming (PPOPP), ACM (2006) 198–208
4. Kotselidis, C., Ansari, M., Jarvis, K., Lujan, M., Kirkham, C., Watson, I.: DiSTM: A software transactional memory framework for clusters. In: Proc. of the International Conference on Parallel Processing (ICPP). (2008) 51–58
5. Bocchino, R.L., Adve, V.S., Chamberlain, B.L.: Software transactional memory for large scale clusters. In: Proc. of the Symposium on Principles and Practice of Parallel Programming (PPOPP), ACM (2008) 247–258
6. Carvalho, N., Cachopo, J., Rodrigues, L., Rito Silva, A.: Versioned transactional shared memory for the FenixEDU web application. In: Proc. of the Workshop on Dependable Distributed Data Management (WDDDM), ACM (2008)

7. Pedone, F., Guerraoui, R., Schiper, A.: The database state machine approach. Distributed and Parallel Databases **14**(1) (2003) 71–98
8. Kemme, B., Alonso, G.: A suite of database replication protocols based on group communication primitives. In: Proc. of the International Conference on Distributed Computing Systems (ICDCS), IEEE Computer Society (1998) 156
9. Patino-Martínez, M., Jiménez-Peris, R., Kemme, B., Alonso, G.: Scalable replication in database clusters. In: Proc. of the International Conference on Distributed Computing (DISC), Springer-Verlag (2000) 315–329
10. Defago, X., Schiper, A., Urban, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Computing Surveys **36**(4) (2004) 372–421
11. Romano, P., Carvalho, N., Rodrigues, L.: Towards distributed software transactional memory systems. In: Proc. of the Workshop on Large-Scale Distributed Systems and Middleware (LADIS). (2008)
12. Cachopo, J., Rito-Silva, A.: Versioned boxes as the basis for memory transactions. Sci. Comput. Program. **63**(2) (2006) 172–185
13. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley (1987)
14. Guerraoui, R., Rodrigues, L.: Introduction to Reliable Distributed Programming. Springer (2006)
15. Miranda, H., Pinto, A., Rodrigues, L.: Appia, a flexible protocol kernel supporting multiple coordinated channels. In: Proc. International Conference on Distributed Computing Systems (ICDCS), IEEE (2001) 707–710
16. Amir, Y., Danilov, C., Stanton, J.: A low latency, loss tolerant architecture and protocol for wide area group communication. In: Proc. of the International Conference on Dependable Systems and Networks (DSN). (2000)
17. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: Proc. of the Conference on the Management of Data (SIGMOD), ACM (1996) 173–182
18. Carvalho, N., Pereira, J., Rodrigues, L.: Towards a generic group communication service. In: Proc. of the International Symposium on Distributed Objects and Applications (DOA). (2006)
19. Couceiro, M., Romano, P., Carvalho, N., Rodrigues, L.: D^{2}STM: Dependable distributed software transactional memory. Technical Report 30/2009, INESC-ID (2009)
20. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7) (1970) 422–426
21. Broder, A., Mitzenmacher, M.: Network Applications of Bloom Filters: A Survey. Internet Mathematics **1**(4) (2003) 485–509
22. Perez-Sorrosal, F., Patino-Martinez, M., Jimenez-Peris, R., Kemme, B.: Consistent and scalable cache replication for multi-tier J2EE applications. In: Proc. of the International Conference on Middleware (Middleware), Springer-Verlag (2007) 328–347
23. Bertsekas, D.P., Tsitsiklis, J.N.: Introduction to Probability. Athena Scientific (2002)
24. Herlihy, M., Luchangco, V., Moir, M.: A flexible framework for implementing software transactional memory. SIGPLAN Not. **41**(10) (2006) 253–262
25. Guerraoui, R., Kapalka, M., Vitek, J.: STMBench7: a benchmark for software transactional memory. SIGOPS Oper. Syst. Rev. **41**(3) (2007) 315–324

SEEDS: The Social Internet Feed Caching and Dissemination Architecture

Ana Nunes[1], José Marques[2], and José Pereira[3]

[1] University of Minho aln@lsd.di.uminho.pt
[2] University of Minho trofa@lsd.di.uminho.pt
[3] University of Minho jop@di.uminho.pt

Abstract. Syndicated content in the Internet has been a huge success ever since the early days of RSS 0.9 and MyNetscape. Currently, it is the cornerstone of content push, ranging from podcasts to emerging Web 2.0 sites such as FriendFeed and Plexus. Unfortunately, the simple technology that makes publication and subscription very simple and flexible, thus explaining in part its success, is also limiting its usefulness in more demanding applications.
This paper proposes a novel distributed architecture for feed caching and dissemination. It leverages social networks as promoters of discovery and aggregation, and peer-to-peer protocols for content distribution, while providing an evolutionary upgrade path that does not disrupt current infrastructure or require changes to publishers' or consumers' habits.

1 Introduction

The Web has grown tremendously and content has evolved to be much more dynamic than in its early days. While some websites still feature essentially static content, others are updated daily, and in some cases (e.g., news oriented websites) the update rate is even higher, amounting to several updates a day.

Consider, for instance, that one tries to stay current with the content of a given website. It would be necessary to keep polling each of its pages to check for updates. If we also consider that instead of one client, a website will typically serve many and take into account the mere size of the Internet, bandwidth consumption and increased server load issues clearly emerge, as the number of clients (and therefore, the polling rate) go up. For example, according to Facebook statistics[4], more than 100 million users log on to Facebook each day.

Syndication[5] has changed the standard paradigm in order to cope with these events, by addressing the problem of knowing when an update is available. This is achieved in a content-push model, based on the publish/subscribe paradigm. Content publishers provide a Web feed: a XML file containing headlines and descriptions, which provides

[4] http://www.facebook.com/press/info.php?statistics

[5] Web syndication refers to how feeds are made available from a website, which is analog to how syndication works in broadcasting. For more information refer to http://www.internetcontentsyndication.org/downloads/whitepapers/content_creation.pdf

a summary of recently added or updated content, that users can subscribe to. However, the illusion of content-push is accomplished using hidden polling.

Web feed technology eases polling by collecting short descriptions of recent updates, with pointers, in a single file that can be polled more easily. Nonetheless, consider the following example. Each Wired Top Stories feed is sized at about 79KB, and from data available at Bloglines[6] it's subscribed to by at least 94,951 users. If separately polling the feed with an estimated refresh period of 2 hours, meaning 12 updates per day, daily polling-related traffic reaches 87.9 GB.

Current protocols such as RSS [1] and Atom [2] allow anyone to easily publish or subscribe to feeds by using simple tools. These protocols prove quite adequate for the large majority of feeds, featuring few subscribers and low data throughput. However, for high throughput or highly popular feeds, the polling mechanism becomes expensive for the publisher [3], making them less adequate.

Bandwidth consumption is aggravated by the generalized lack of support in both servers and clients for the retrieval of just new feed entries. Also, once a user subscribes to a feed, she'll likely maintain it for a long time and stemming from the use of feed readers and from the global nature of the Internet, polling will occur persistently. Some techniques have been proposed to try to minimize this problem [4], although with limited success.

Web feeds are also a major feature of social networking services, where they're being used to disseminate frequently updated information to interested users. For instance, Facebook uses feeds to highlight changes in social circles and also to disseminate information about any particular user; FriendFeed [7] condenses the content shared on multiple services by a group of selected users into a feed.

In order to mitigate bandwidth consumption at the publisher while achieving timely content delivery for the subscriber, we present an approach to feed dissemination using gossip in peer-to-peer groups. By leveraging social networks to aid in defining these groups, we address the issue of reducing the overhead associated with a generic gossip strategy and also derive additional functionality in the form of content discovery.

The rest of the paper is structured as follows: Section 2 describes the current feed dissemination architecture in detail. Section 3 states the goals of the current proposal, and in Section 4, an architecture is proposed to achieve these goals. Section 5 introduces the current prototype implementation. Section 6 compares this approach with previous work and Section 7 concludes the paper, underlining current research directions.

2 Background

Fig. 1 depicts the common Internet Web feed architecture. Web feeds are XML files composed of entries, which may be headlines, full-text articles, excerpts, summaries, and/or links to content on a website, along with metadata. RSS and Atom are XML-based document formats, both used for feeds. For futher details on these concepts and on how they relate to each other see [5] and/or [6]. Feeds can be subscribed to directly

[6] `http://bloglines.com`, a popular Web based feed reader that provides usage statistics.
[7] `http://friendfeed.com`

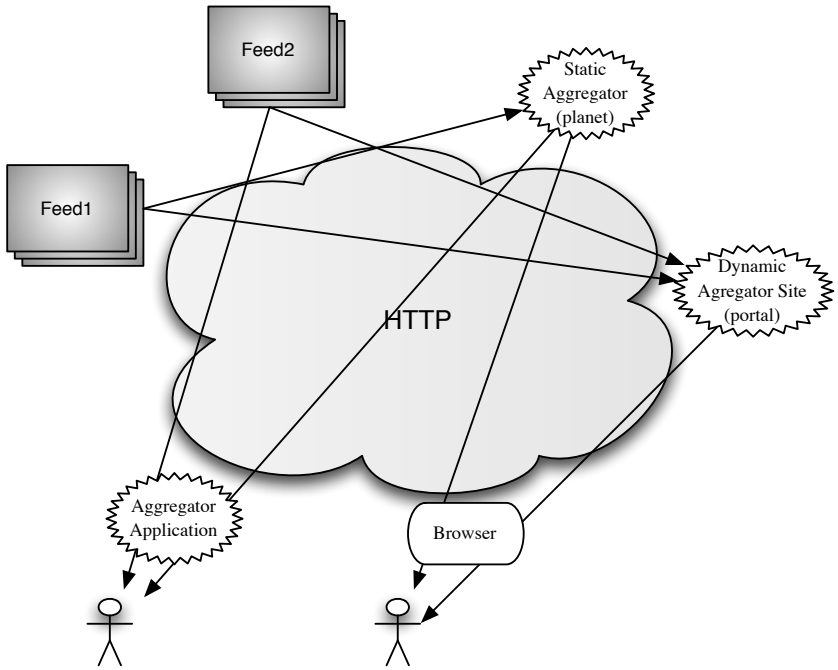


Fig. 1. Internet Web feed architecture.

by the end-user with an aggregator application, through a portal (the original intent), or by a planet. Aggregators, also known as feed readers or news readers, are applications that fetch feeds, usually in an automated manner, and present entries to the user (e.g. RSSOwl). Portals are websites featuring content from a static set of chosen feeds. Planets are websites featuring aggregated content of interest to a particular community of users. Notice that planets are a manual effort to identify communities with shared interests and provide them with commonly accepted content. Portals are much less prone to allow that sharing. On the other hand, planets don't allow any customization of the feeds by the end user. All aggregators perform caching, but only in portals and planets is the cache shared between multiple users.

Nowadays, Web feeds are being used to convey various types of content: from news headlines, to podcasts and video podcasts, to being a feature in Web 2.0 services. As an example from social networking services, more than 1 billion pieces of content (Web links, news stories, blog posts, notes, photos, etc.) are shared each week in Facebook, which are then published in Web feeds.

3 Goals

The proposed architecture aims at achieving the following goals:

Multiple Dissemination Protocols Content type and traffic patterns vary widely. For example, feeds can be mainly composed of text, normally for website related news, or feature richer content (feeds with audio/video attached), generating traffic with different characteristics. Broadcatching, which refers to automatic downloading of content published in feeds (e.g. TV series related feeds), imposes yet another traffic pattern. The number of participants (essentially subscribers) should also be considered. It makes sense not to use the same dissemination protocol for such different cases, so it is necessary to differentiate them and choose the right protocol. For example, small text based entries with a very high update rate could be disseminated via NeEM [7] and use BitTorrent [8] for audio and video content. This way, we are able to disseminate small content quickly and large content efficiently.

No Change to the Source One of the reasons why widespread adoption of other feed dissemination proposals (e.g. FeedTree [9]) hasn't occurred was that they relied heavily on the source to explicitly join the system. We should not expect the source to collaborate. Also, it should be possible to use the existing architecture and protocols without fundamentally altering them for this purpose.

No Change to the Target Clients shouldn't need to change their habits and be able to use any aggregator application to interact with the system, thus removing one of the biggest barriers for the adoption of this proposal. Also, some content dissemination frameworks require a specific client application, and some support only specific-purpose social networks (eg. Tribler [10]). It should be possible to use pre-existent social networks (e.g. Hi5 [8]) and take advantage of different dissemination architectures without requiring a special client for each one, selecting the most appropriate method automatically.

Enhanced Discovery and Aggregation We enhance the subscribers' experience with recommendations of new entries or feeds. These could be content-related or stem from social connections.

4 Architecture

As stated in the previous section, it is our goal to keep changes to a minimum for all parties in the current Web feed architecture. To comply with the no-changes guideline, the interface to the proposed system is simply the subscriber's news aggregator application of choice. To this end, we provide a proxy server to be deployed locally, at the client, which intercepts and interprets the subscriber's requests. The subscriber only needs to configure a third-party aggregator application to use the local proxy server.

Fig. 2 shows the proposed SEEDS architecture.

[8] `http://www.hi5.com`

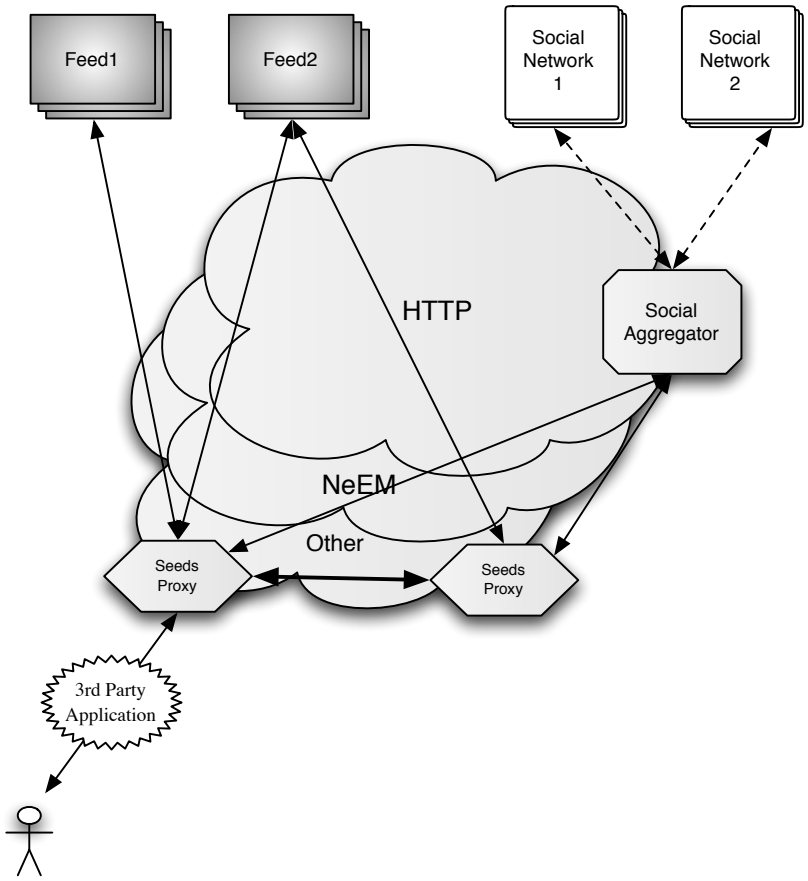


Fig. 2. SEEDS distribution and caching architecture.

The proxy identifies a feed request from a third-party application and either returns the corresponding cached feed, or requests it from the publisher. In case the feed is hitherto unknown, the proxy contacts a social aggregator to obtain instructions on how to handle the feed's caching and dissemination; if none is available, a local configuration is applied.

SEEDS proxies form peer-to-peer groups, translated into overlay networks. A multicast protocol (e.g. NeEM) instance is deployed in each group for entry dissemination. If deemed appropriate, other protocols can also be deployed. The feed-to-group mapping is defined by social aggregators, which gather information from social networking services along with perceived feed popularity. This information is also leveraged to provide Web feed recommendations. Social aggregators are separate, independently-deployed

modules. Each SEEDS proxy also provides a locally generated recommendation feed, which can easily be subscribed to by the user or simply ignored if desired.

Because of the overhead in group formation and maintenance, only popular feeds are allocated to groups. Feeds featuring few subscribers are simply cached locally, but its entries are not disseminated. The popularity threshold is determined by the social aggregator and dynamically adjusted.

Social aggregators can be promoted by social networking services as a service to its users. Thus, different services deploy different aggregators, leading to different group mappings for the same feeds.

5 Implementation

To illustrate our architecture, we developed a software prototype that implements the features outlined in the last section. The SEEDS architecture advocates a two module configuration. A proxy is deployed in the subscribers machine, which connects to a social aggregator on the internet. We examine each of these modules in detail.

5.1 Seeds Proxy

To ease the process of using SEEDS, the SEEDS proxy is provided as a Java WebStart application. The user can thus deploy it on her machine by, e.g. following a link in a social networking site of her choosing. This proxy is then able to use UPnP to configure itself in most home networking environments, even when behind firewalls.

The SEEDS proxy internal architecture is shown in Fig. 3. The proxy is composed of three main modules: the HTTP Servlet which interprets the requests received by third-party applications and publishes a local recommendation feed; the Cache Manager which handles feed caching, dissemination to groups and recommendation feed generation; and an HTTP Client, responsible for fetching feeds from the publishers and interacting with social aggregators to, among other interactions, obtain feed-related configuration.

The issue of discovering a social aggregator's URL is outside the scope of this work. Here is a step-by-step feed retrieval example:

1. A third-party aggregator application makes a feed request.
2. The request is interpreted by the HTTP Servlet module to check whether the URL effectively points to a feed and if so passes it on to the Cache Manager module.
3. The Cache Manager module does a lookup for the URL in the cache. If the URL is being cached, it retrieves the cached feed; otherwise, the HTTP Client retrieves the feed from the publisher and its configuration from a social aggregator. Then, the Cache Manager module deploys the dissemination protocol module, configuring it with a peer-to-peer group according to the configuration information it received. The relation between feeds and groups is many-to-one.
4. In the background, the Cache Manager is both sending updates to peer-to-peer groups and listening for other updates from them.

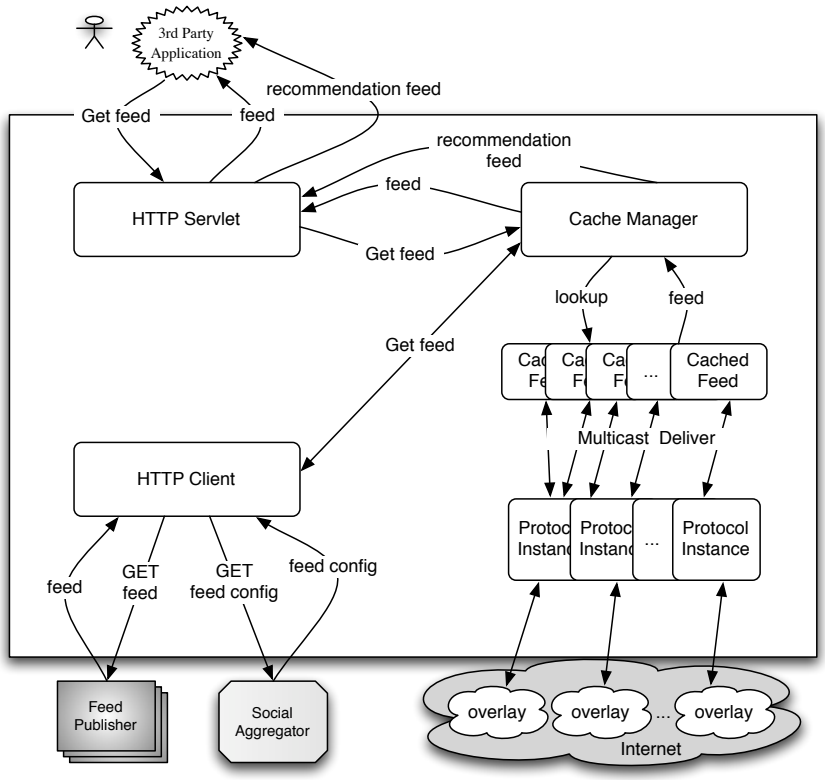


Fig. 3. SEEDS proxy implementation.

Notice that interaction with the social aggregator is only necessary when the proxy is presented with a new feed. Therefore if no social aggregator can be contacted, existing groups and known feeds are not affected. New feeds that are subscribed to in this situation, or for which the social aggregator simply does not provide a group, are cached locally, without dissemination, periodically refreshed from the publisher.

Although both the SEEDS architecture and the proxy's implementation are dissemination protocol agnostic, currently, we provide only an epidemic multicast protocol module. Epidemic multicast[11, 12], also dubbed probabilistic or gossip-based, is based on the simple procedure of relaying each received item to a small random subset of other nodes. This mechanism ensures these protocols are well suited to disseminate events to a large number of interested parties, while achieving stable high throughput, and coping with node or network faults. It has been shown that the probability that all nodes are informed can be made as large as desired, without reaching 1, and that messages spread exponentially fast. NeEM[7] is an implementation of epidemic multicast in wide-area

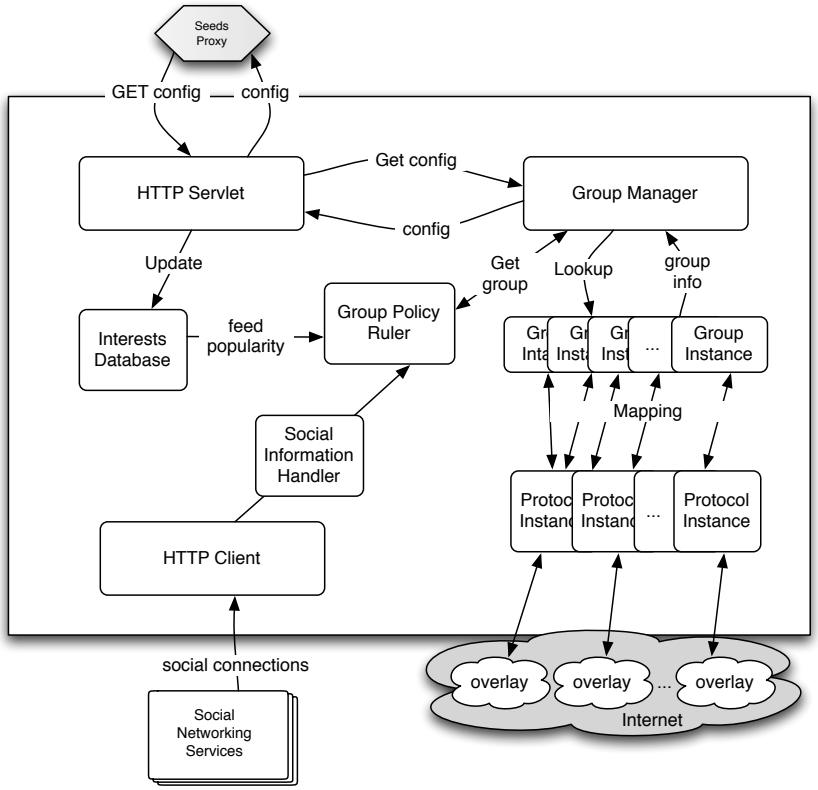


Fig. 4. SEEDS social aggregator server implementation.

networks relying on connection-oriented transport connections and on specific mechanisms to avoid network congestion. The resulting overlay network is automatically managed by the protocol. Its scalability and reliability make this protocol ideal for disseminating feed entries in groups where most peers are interested in the same set of feeds, and with high update rates. A protocol directed at a very large number of feeds with a small number of interested parties should also be provided as in FeedTree [9].

The recommendation feed is generated by the Cache Manager by collecting the feed URLs that are being disseminated in the groups but that are not yet subscribed to by the user and made available through the HTTP Servlet.

The SEEDS proxy implementation can be found at `http://seeds.sf.net`.

5.2 Social Aggregator

The social aggregator module, is presented in Fig. 4. Here, there are six main modules: the HTTP Servlet which receives group requests from SEEDS proxies; the Group

Manager which creates and manages dissemination groups; the Group Policy Ruler that determines feed-to-group mappings; the Interests Database which tallies feed configuration requests; the Social Information Handler which handles social networking and bookmarking information; and the HTTP Client which interfaces with social networking services. Both the Interests Database and the Social Information Handles support the Group Policy Ruler.

Here is a step-by-step group retrieval for a feed:

1. The HTTP Servlet interprets the configuration request and passes it on to the Group Manager.
2. The Group Manager queries the Group Policy Ruler for a feed-to-group mapping. The Group Policy Ruler considers both the data present at the Interests Database and the information afforded by the Social Information Handler to decide on the most convenient mapping. In fact, the Group Policy Ruler might not assign a group to the feed, if it finds that the its low popularity doesn't warrant it. The relation between groups and protocol instances is one-to-one.
3. Finally, the Group Manager returns a configuration stating the dissemination protocol to be used, the group to which the feed belongs to and other relevant parameters.

The Interests Database provides the Group Policy Ruler with accurate data related to the number of subscribers for each feed, enabling an efficient management of the dissemination groups.

The social networking information gathered by the Social Information Handler makes it possible to create groups according to social connections. In some social networking services, such as FriendFeed, social connections represent a similarity of interests. Therefore, the feeds mapped to groups formed in this manner should be of interest to the majority of participants. Besides reducing the overhead of disseminating entries to non-interested users, this mechanism also provides a source for recommendation: the feeds that are disseminated in a group but that are not yet subscribed by a given user will probably be of interest to him. Also, the tagging information found at social bookmarking services might be leveraged to categorize either feeds or specific entries in such a way that recommendations of related content can be generated.

Currently, feed-to-group mappings are statically determined, with a configuration drawn from existing Planets. Work is in progress to do it dynamically and provide a full-featured implementation of a social aggregator.

6 Related Work

There are a number of technologies that approximate either the caching and dissemination or the social aggregation and discovery aspects of SEEDS. For instance, Planet sites are used to aggregate syndicated Web content for online communities and to display it on a single page. There is, without question a social component to this, albeit a static one, as viewers do not control which feeds are aggregated. Also, for different areas of interest, a client has to visit different planets which impose sharp boundaries on communities.

FeedBurner [13] provides several services to feed publishers. To do so, it works like a centralized cache, in that the "burned" version of the feed is the one publicized. The original feed is checked for updates every 30 minutes. Although server load is moved away from the publisher's server to FeedBurner's servers, the bandwidth consumption problem remains the same. Also, the published "burned" feed's timeliness is limited.

FeedTree [9] is an approach directed at collaborative micronews delivery, and thus the closest to our proposal in terms of dissemination strategy. It does however use a single communication protocol (Scribe [14]) built on top of a peer-to-peer overlay network (Pastry [15]) and does not promote discovery and aggregation based on interests or social connections.

Tribler [10] is an open source BitTorrent client that leverages a social network to provide content recommendations and cooperative downloading, using a mechanism of bandwidth donation. Tribler users can create profiles, groups, add friends and tag content. Tribler's decentralized recommendation uses an algorithm based on an epidemic protocol, by exchanging download histories either with peers known to have similar tastes but also with peers chosen at random. Tribler is geared towards file-sharing, particularly video content. However, Tribler is focused only on multimedia content and requires custom tools to participate.

Last.fm [16] is a service that can be used for listening to music and that provides content recommendation based on user's profiles. When a user listens to a music, this information is added to a profile page, visible to other users. With this data, Last.fm suggests music to users, also allowing them to create personalized radio stations. In this social network, the notion of friendship exists along side the notion of neighborhood, taken from the similarity in musical taste. This service is designed to work with musical content only and is restricted to centrally supplied content.

FeedEx[17] is a news feed exchange system, where peers cooperate by exchanging feed documents with their neighbours. In FeedEx, content recommendation is provided, stemming from two sources: from entries rating (or votes) or by viewing a read as an implicit endorsement; from the feed subscription sets, determining similar interests. Neighbour selection is done at each peer, based on the overlap in subscription sets, considering also other factors like topological proximity.
FeedEx does provide interesting features, such as the construction of a distributed news archive and an incentive mechanism to minimize selfish behaviour. But, in this proposal, a specifically tailored peer-to-peer protocol is used, and it is heavily intertwined with the application itself. Thus, FeedEx does not support different dissemination protocols.

Content distribution networks attempt to optimize content delivery, implementing Web caches to store popular content closer to the user, thus reducing bandwidth requirements and server load, increasing reliability and also improving client response times. For example, the Coral Content Distribution Network [18] is an academic, free, peer-to-peer content distribution network that leverages the bandwidth of participating nodes as proxy servers, making it similar to a distributed Web proxy. However, network nodes are not users and there is no social component associated.

7 Conclusions and Future Work

In this paper we have introduced the architecture and the initial prototype implementation of SEEDS, an evolutionary proposal for Internet feeds infrastructure. While no changes are required to either the subscriber or the publisher, it should reduce server bandwidth usage when a large number of subscribers choose to participate. In contrast with previous proposals, SEEDS provides the means to foster the participation of such large number of participants by leveraging social networking and bookmarking sites. Finally, information that can be obtained from tagging performed within social bookmarking services can be used for group policy and suggestions. A full-featured implementation of the social aggregator is to be developed. Also, the issue of discovering social aggregators is to be addressed.

The current SEEDS prototype enables several interesting research problems. First, we are integrating SEEDS with a social networking service and working on obtaining traffic statistics to validate current hypotheses. On the other hand, we are also working on optimizing the multicast protocol and evaluating different protocols for different traffic patterns.

Finally, it is interesting to speculate whether improving Internet feeds such that (i) a very large number of subscribers to fast changing content can be supported without vast server resources; and (ii) aggregation and discovery is much easier and flexible, will enable radically new uses for this technology.

References

1. Rss protocol (specification). `http://www.rssboard.org/rss-specification`.
2. Atom protocol (ietf draft). `http://bitworking.org/projects/atom/draft-ietf-atompub-protocol-04.html`.
3. Matthew Hicks. Rss comes with bandwidth price tag. http://www.eweek.com/c/a/Messaging-and-Collaboration/RSS-Comes-with-Bandwi dth-Price-Tag/.
4. Randy Charles Morin. Howto rss feed state. http://www.kbcafe.com/rss/rssfeedstate.html.
5. S. Powers. What are Syndication Feeds. O'Reilly, 2005.
6. H. Wittenbrink. RSS And Atom: Understanding And Implementing Content Feeds And Syndication. Packt Publishing, 2005.
7. J. Pereira, L. Rodrigues, M. J. Monteiro, R. Oliveira, and A.-M. Kermarrec. NEEM: Network-friendly epidemic multicast. In SRDS, pages 15–24. IEEE Computer Society, 2003.
8. Bram Cohen. Incentives build robustness in bittorrent, May 2003.
9. D. Sandler, A. Mislove, A. Post, and P. Druschel. Feedtree: Sharing web micronews with peer-to-peer event notification, 2005.
10. J.A. Pouwelse, P. Garbacki, J. Wangand A. Bakker, J. Yang, A. Iosup, D. Epema, M.Reinders, M.R. van Steen, and H.J. Sips. Tribler: A social-based based peer to peer system. In 5th Int'l Workshop on Peer-to-Peer Systems (IPTPS), Feb 2006.
11. J. Pereira, L. Rodrigues, M. J. Monteiro, R. Oliveira, and A.-M. Kermarrec. Neem: Network-friendly epidemic multicast. Reliable Distributed Systems, IEEE Symposium on, 0:15, 2003.
12. P.T. Eugster, R. Guerraoui, A.M. Kermarrec, L. Massoulie, and AJ Ganesh. From epidemics to distributed computing. IEEE Computer, 37(5):60–67, 2004.
13. Feedburner. `http://www.feedburner.com/fb/a/home`.

14. M. Castro, P. Druschel, A.-M. Kermarrec, and A.I.T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. Selected Areas in Communications, IEEE Journal on, 20(8):1489–1499, Oct 2002.
15. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science, 2218, 2001.
16. Last.fm. `http://www.last.fm/`.
17. Seung Jun and Mustaque Ahamad. Feedex: collaborative exchange of news feeds. In WWW '06: Proceedings of the 15th international conference on World Wide Web, pages 113–122, New York, NY, USA, 2006. ACM.
18. M. Freedman, E. Freudenthal, and D. Mazi. Democratizing content publication with coral, 2004.

FT-OSGi: Extensões à Plataforma OSGi para Tolerância a Faltas*

(Resumo Estendido)

Carlos Torrão, Nuno Carvalho, and Luís Rodrigues

INESC-ID/IST
carlos.torrao@ist.utl.pt, nonius@gsd.inesc-id.pt, ler@ist.utl.pt

1 Introdução

A plataforma OSGi [1] foi criada pela organização *OSGi Alliance* e concretiza uma especificação para a composição de serviços que possam ser instalados dinamicamente e que se invocam mutuamente. Esta plataforma foi concretizada para a linguagem Java e fornece primitivas que permitem a construção de aplicações a partir de pequenos módulos reutilizáveis, que colaboram entre si. Muitas áreas aplicacionais que usam a plataforma OSGi têm requisitos de disponibilidade e tolerância a faltas. Na área da domótica foram reportados vários problemas de fiabilidade, que se reflectem na (falta de) satisfação do utilizador final [2]. Assim sendo, é da maior importância desenhar e implementar ferramentas que forneçam tolerância a faltas neste tipo de aplicações, sem, no entanto, perder a compatibilidade entre os vários módulos.

2 Trabalho Relacionado

O núcleo da plataforma OSGi suporta a instalação e remoção, em tempo de execução, de aplicações extensíveis, a que chamamos *módulos*. Uma das principais vantagens da plataforma OSGi é a sua capacidade de instalação em tempo de execução, sendo possível instalar, actualizar, remover, iniciar e parar módulos numa instância OSGi em execução.

Do nosso conhecimento até à data, não existe nenhum trabalho que forneça suporte de tolerância a faltas para a arquitectura OSGi de uma forma genérica e completa, apesar de existirem os seguintes trabalhos nessa direcção: OSGi-based Gateway Replication [3] e Towards Reliable OSGi Framework [4].

3 FT-OSGi

Esta secção apresenta o FT-OSGi, que consiste num conjunto de extensões à plataforma de serviços OSGi. Estas extensões visam melhorar a disponibilidade e fiabilidade de aplicações OSGi, de uma forma transparente e simples.

* Este trabalho foi parcialmente suportado pela FCT através do projecto Pastramy (PTDC/EIA/72405/2006).

Os serviços são replicados em vários nós de um sistema distribuído. O cliente, quando deseja aceder a um serviço, comunica com o grupo de réplicas desse serviço e não com uma única instância. Os serviços podem ter dois modos de funcionamento relativamente ao seu estado interno: com estado ou sem estado. A manutenção de estado replicado requer suporte explícito por parte do programador do serviço. São suportados dois tipos de replicação: replicação activa e replicação passiva.

É também possível definir como são seleccionadas as respostas retornadas ao cliente do serviço replicado. No caso de replicação activa, um pedido é executado em todas as réplicas e o cliente obtém um número de respostas equivalente ao número de réplicas existentes. O intermediário do lado do cliente tem a responsabilidade de filtrar os pedidos de forma a tornar a replicação transparente. O FT-OSGi suporta três modos de selecção de respostas: `primeira-resposta`, `todas-as-respostas`, e `maioria-das-resposta`. No modo `primeira-resposta`, a primeira resposta a ser recebida é logo processada para o cliente, e todas as respostas que lhe seguem são descartadas. No modo `todas-as-respostas`, só é passada uma resposta para o cliente quando todas as respostas foram recebidas e verificadas como sendo iguais. Caso uma ou mais respostas sejam diferentes, é lançada uma excepção com essa informação. Finalmente, o modo `maioria-das-respostas` retorna a resposta maioritária, mesmo que existam respostas divergentes.

Finalmente, visto que os clientes têm uma instância OSGi local, a semântica dos eventos OSGi é mantida, fazendo chegar a todos os clientes notificações dos eventos gerados pelos serviços replicados, assim como eventos acerca do ciclo de vida destes serviços. Para mascarar a replicação, um intermediário do lado do cliente é responsável por filtrar as cópias do mesmo evento recebidas pelo cliente (usando algoritmos semelhantes aos usados para processar respostas aos pedidos).

Sendo o FT-OSGi uma extensão para a plataforma OSGi, existe uma instância OSGi em execução em todos os nós do sistema. Para além disso, o FT-OSGi combina componentes desenvolvidos por terceiros com componentes desenvolvidos pelos autores para suportar as funcionalidades acima descritas. Os componentes desenvolvidos por terceiros são o R-OSGi e o serviço de comunicação em grupo, que se descrevem de seguida.

O R-OSGi [5] consiste numa plataforma capaz de distribuir uma aplicação OSGi por vários nós de uma rede. A camada R-OSGi assenta sobre a arquitectura OSGi original de uma forma transparente para as aplicações, sendo possível colocar qualquer aplicação que assente sobre OSGi a funcionar sobre o R-OSGi. O R-OSGi baseia-se no uso de intermediários (que representam um serviço numa máquina remota) e no uso do protocolo *Service Location Protocol* (SLP) [6]. Para cada serviço publicado por um módulo OSGi num nó específico, quando é necessário num outro nó, é criado um intermediário que representa o serviço remotamente. Este intermediário simula localmente o serviço que está no nó remoto, efectuando invocações remotas através da rede. O SLP é usado para permitir a descoberta dos serviços na rede.

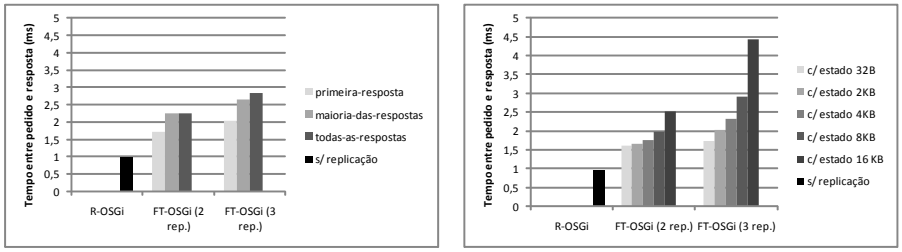
O FT-OSGi usa um serviço genérico de comunicação em grupo (jGCS) [7] que suporta o uso de várias concretizações. Neste caso foi concretizado usando o Appia [8]. O Appia permite a comunicação entre de elementos fora de um grupo com elementos do grupo.

Sobre os serviços acima descritos, foram desenvolvidos os seguintes componentes: *FT Service Handler* e *FT-Core* que, no seu conjunto, permitem fornecer tolerância a faltas a serviços OSGi. O *FT Service Handler* é responsável por obter e disponibilizar a informação relativa aos serviços remotos que podem ser acedidos através da instância local de OSGi. Em particular, as propriedades dos serviços registam as opções de configuração do FT-OSGi (tal como o tipo de replicação utilizada, a forma como se processam as respostas, etc). O *FT-Core* é responsável por manter a coerência das réplicas que se executam em diferentes instâncias de OSGi e por esconder dos clientes a complexidade associada a esta tarefa. O componente *FT-Core* encontra-se dividido em sub-componentes, que serão descritos de seguida. O *Appia Channel Factory* é o componente responsável pela definição do serviço de replicação. O grupo de comunicação suporta dois fluxos: a comunicação entre réplicas e a comunicação entre o cliente e o grupo de réplicas. Para cada um destes fluxos de comunicação, são criados canais Appia. O canal de comunicação entre réplicas é criado quando é registado um serviço com propriedades de tolerância a faltas numa instância de OSGi (num servidor), sendo concretizado pelo componente *Appia Channel (Server)*. O canal de comunicação do cliente para o grupo é criado quando esse cliente deseja invocar o serviço replicado e é concretizado pelo componente *Appia Channel (Client)*. O componente *Replication Mechanisms* é responsável pela gestão da replicação usada nos serviços. Este componente implementa as duas técnicas de replicação suportadas (activa e passiva), tanto para serviços com estado como para serviços sem estado. Sendo também responsável pela recuperação de faltas e pela transferência de estado a novos elementos que são adicionados ao grupo.

4 Avaliação

Para efectuar a avaliação do FT-OSGi foram utilizados testes que executam várias invocações de um método vazio, sem processamento, que recebe 4 KBytes de dados como argumento. Para estes testes, medimos os tempos de resposta, para a versão centralizada usando R-OSGi e para a versão replicada com o FT-OSGi.

Através dos resultados obtidos (ver Figura 1) foi possível extrair as seguintes conclusões. Enquanto a execução remota de um serviço não replicado só exige dois passos de comunicação (pedido e resposta), a execução de um serviço replicado exige pelo menos quatro passos de comunicação (uma vez que inclui o custo adicional da coordenação entre as réplicas) e, no caso da replicação passiva, a transferência de estado entre a réplica primária e as réplicas secundárias, o que justifica as diferenças de desempenho observadas. Pode também observar-se que quanto maior for o número de réplicas, maior é o custo no desempenho.



(a) Replicação activa (b) Replicação passiva

Figura 1. Custo adicional da replicação.

5 Conclusões

Este artigo descreve e avalia o FT-OSGi, um conjunto de extensões à plataforma OSGi orientada a fornecer tolerância a faltas. No FT-OSGi, cada serviço pode ser configurado para usar replicação passiva e activa, assim como propriedades de QoS. Apesar da replicação induzir um custo adicional na invocação dos serviços, para números reduzidos de réplicas os valores são aceitáveis para muitas aplicações.

Agradecimentos: Os autores agradecem ao João Leitão pelos comentários feitos a versões preliminares deste artigo.

Referências

1. OSGi Alliance: Osgi service platform core specification (April 2007) http://www.osgi.org/Download/Release4V41.
2. Kaila, L., Mikkonen, J., Vainio, A.M., Vanhala, J.: The ehome - a practical smart home implementation. In: Proceedings of the workshop Pervasive Computing @ Home, Sydney, Australia (May 2008)
3. Thomsen, J.: Osgi-based gateway replication. Proceedings of the IADIS Applied Computing Conference 2006 (2006) 123–129
4. Ahn, H., Oh, H., Sung, C.: Towards reliable osgi framework and applications. Proceedings of the 2006 ACM symposium on Applied computing (2006) 1456–1461
5. Rellermeyer, J., Alonso, G., Roscoe, T.: R-osgi: Distributed applications through software modularization. Middleware (2007) 1–20
6. Guttman, E.: Service location protocol: automatic discovery of ip network services. Internet Computing, IEEE **3**(4) (Jul/Aug 1999) 71–80
7. Carvalho, N., Pereira, J., Rodrigues, L.: Towards a generic group communication service. In: Proceedings of the 8th International Symposium on Distributed Objects and Applications (DOA), Montpellier, France (Oct 2006)
8. Miranda, H., Pinto, A., Rodrigues, L.: Appia, a flexible protocol kernel supporting multiple coordinated channels. In: Proceedings of the 21st International Conference on Distributed Computing Systems, Phoenix, Arizona, IEEE (Apr 2001) 707–710

Capi: Cloud Computing API

Bruno Costa and Miguel Matos and António Sousa
{blc,mm}@lsd.di.uminho.pt, als@di.uminho.pt

Universidade do Minho, Braga, Portugal

Abstract. Cloud Computing is an emerging business model to provide access to IT resources in a pay per use fashion. Those resources range from low-level virtual machines, passing by application platforms and ending in ready to use software delivered through the Internet, creating a layered stack of differentiated services. Different cloud vendors tend to specialize in different levels, offering different services, each with its own proprietary API. This clearly led to provider lock-in and hinders the portability of a given application between different providers. The goal of this paper is therefore to provide an abstraction of the different levels of services in order to reduce vendor lock-in and improve portability, two issues that we believe impair the adoption of the Cloud Computing model.

1 Introduction

Cloud Computing's [9] key idea is to provide access to arbitrary resources over the Internet which will be otherwise confined to specialized infrastructures. The access to the Cloud is based on a pay per use model, providing resources that range from the hardware level, to software as a service, passing by an intermediate level where applications could be deployed.

Although there are already multiple providers world-wide [3,1,6] providing services at each level of the Cloud stack, there are no standard interfaces to access those services. This lack of standardization leads to vendor lock-in and decreases portability as applications need to be (re-)written to comply with a given API, made available by the Cloud service provider.

In this paper, we propose a general purpose Cloud API (CAPI), that offers programming interface abstractions to each one of the levels of the Cloud stack. With this general API, customers are able to build their applications against it and then, by recurring to the concrete implementation of the provider - the driver - use the underlying services. Additionally, CAPI considers all the abstraction levels, providing the customer with a common interface to manage all of them, using different Cloud providers, at different abstraction levels, in a transparent fashion.

As can be seen in Figure 1, the different abstraction levels of the Cloud stack can be built using the functionalities of the level below it.

In the lowest abstraction level, Infrastructure as a Service (IaaS), the providers offers virtualized hardware solutions, such as storage and processing capacity

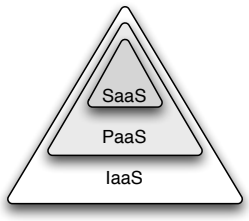


Fig. 1. Cloud services levels.

(e.g. Amazon EC2 [1]). The Platform as a Service (PaaS) level models a software stack - the platform - that could be used to develop, deploy and run applications (e.g. Google App Engine [3] and Force.com [8]). In the less featured level, only a service is provided to the client, which is a remote application available as a web service or as a browser enabled application (e.g. Yahoo! Mail [10] and Google Docs [4]).

The rest of this paper is structured as follows. In Section 2 we present the architecture of the API and the rationale that led to it. Finally Section 3 concludes the paper.

2 Architecture

Taking into account the functional requirements of the system it is clear that we need a modular, pluggable mechanism in which it is possible to control the modules' life-cycle. To fulfill these requirements, our proposal is built atop the OSGi [7] platform. The dynamic properties of OSGi enables partial deployment of modules, as well as changing them at runtime.

In order to enable the seamless integration with management user interfaces, CAPI exports its interfaces through standard Java Management eXtensions (JMX) [5].

Figure 2 depicts the CAPI architecture. It mainly consists in three modules that represent each one of the abstraction levels in the Cloud stack, and an additional module that exposes the functionalities of the other modules through JMX. In addition to these, Monitoring and Security capabilities appear as pervasive to all abstraction levels.

As the API is intended to be general we have to model abstract concepts into the definition of the interface. The rationale behind the proposal is to think in terms of entities, their relationships and the expected behaviour.

2.1 IaaS: Resource Deployment

In the lowest abstraction level we model everything as a *Resource*, either it is a hard disk, a CPU, storage and so on. The basic idea is to manipulate all the

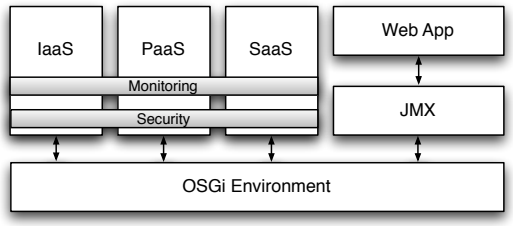


Fig. 2. Architecture Overview.

components available at this level through a common interface with well defined properties.

In this module we define three important interfaces. The *Resource Pack* models image bundles that can be deployed to the provider that includes pre-built images and custom-made images. A *Resource Container* is an abstraction of a running *Resource Pack*, having the ability to add and remove resources at runtime. The *Resource Manager* appears as an entry point to this module, offering capabilities of management and monitoring to the different resources deployed.

In addition to these, resources deployed to the Cloud usually have an internal network encompassing the resources containers of a given customer. This is managed by a *Network* interface, that offers the ability to expose the machines in the internal network to the public network.

2.2 PaaS: Applications Management

This middleware layer support an abstraction for running applications on the Cloud provider platform. In the PaaS you develop applications that can be deployed in the Cloud, following the software development kit offered by the provider. Therefore, we see the applications in the cloud as a set of *Managed Applications* that we are able to manipulate.

The *Managed Applications* have a well defined lifecycle. Thus, it is possible to start and stop that application, as well as observe its state. It also permits to watch each application state and invoke runtime operations exposed by it. To complete the module, the *Platform Manager* controls the bootstrap of all the applications.

2.3 Software as a Service (SaaS): Monitoring Services

The conceptual idea is to provide monitoring capabilities to the users of SaaS. The restriction of functionalities available in this module is directly related to its low flexibility. In this level, the provider offers a complete software solution through the Internet, usually accessible to the customer via a web browser.

All components in SaaS are seen as *Monitored Services* which state we can monitor.

3 Conclusion

In this paper we presented CAPI, a general-purpose Cloud Computing API that covers the different abstraction levels of Cloud services available today. We took an high level approach that focus in the management issues raised when managing components, be it virtual machines or software, in a Cloud environment. With this approach our proposal is able to cope with the offerings of the different providers at each level, and with the imposition of the different abstraction levels on top of the lower ones.

Due to its flexible nature it is possible to build applications that use services from different cloud providers and at different abstraction levels, effectively increasing the portability among different Cloud providers and therefore breaking the lock-in to a specific vendor.

As future work we intend to leverage on the JMX-exposed interfaces to provide a friendly and easy to use web based interface. As the Cloud platform evolves and matures, new abstractions and functionalities will surely arise and thus CAPI must be constantly improved to cope with the new cloud requirements.

References

1. Amazon.com, Inc. Amazon elastic compute cloud. http://aws.amazon.com/ec2, 2009.
2. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.
3. Google. App engine. http://code.google.com/appengine, 2009.
4. Google. Google Docs. http://docs.google.com/, 2009.
5. E. McManus. Java management extensions. Technical report, Java Community Process (JSR'3), 2006.
6. Microsoft Corporation. Azure services platform. http://www.microsoft.com/azure/default.mspx, 2008.
7. OSGi Alliance. Osgi service platform. http://osgi.org/osgi_technology/download_specs.asp, 2005.
8. salesforce.com. Force.com. http://www.salesforce.com/force/, 2009.
9. L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.
10. Yahoo! Inc. Yahoo! Mail. http://mail.yahoo.com, 2009.

Sessão 4C: Multi-track

Encontro de Negócios Digitais

COMUNIDADES VIRTUAIS AO SERVIÇO DO ENSINO

Vitor Santos[1], Luís Amaral[2]

[1] Microsoft, Lisboa, Portugal

[2] Universidade do Minho, Guimarães, Portugal

vitors@microsoft.com, amaral@bragatel.pt

Resumo. Neste artigo abordamos a problemática das comunidades virtuais reunindo um conjunto de argumentos que defendem que estas podem dar um importante contributo para o ensino. A base da argumentação estrutura-se nas características sociológicas e tecnológicas das comunidades virtuais e nos modelos de aprendizagem construtivistas.

Palavras Chave: Comunidades virtuais, Comunidades de aprendizagem

1 Introdução

As Comunidades Virtuais são hoje um fenómeno social na internet. Não existe um consenso quanto a uma definição precisa do que é uma comunidade virtual. Para Rheingold (Rheingold, 1996), destacando um dos aspectos mais evidentes, seria formada por um ecossistema de sub-culturas, e possuiria a característica semelhante a uma espécie de colónia de micro organismos em constante ebulição.

Ainda segundo o mesmo autor (Rheingold,1996, p.18): "as comunidades virtuais são agregados sociais nascidos na "Rede" quando os intervenientes de um debate o levam por diante em número e sentimento suficientes para formarem teias de relações pessoais no ciberespaço".

Deste conceito, podemos destacar alguns pontos que parecem reflectir os elementos básicos apontados por Palacios (Palacios, 1995) na formação do conceito de comunidade: O sentimento de pertença; A territorialidade (geográfica e/ou simbólica); A permanência; A ligação entre sentimento de comunidade; O carácter cooperativo e emergência de um projecto comum; A existência de formas próprias para a comunicação; A tendência à institucionalização.Lévy (Lévy, 1999, p. 127) salienta, para além dos pontos já referidos, a construção de um projecto comum como elemento agregador e potenciador das dinâmicas sociais constituídas nestes espaços de convívio: "Uma Comunidade Virtual é construída sobre as afinidades de interesses, de conhecimentos, sobre projectos mútuos, num processo de cooperação ou de troca, tudo isso independentemente das proximidades geográficas e das filiações institucionais.".

A interacção entre os participantes nas comunidades virtuais é feita através de comunicação localizada num espaço deslocalizado territorialmente, sem que existam, normalmente, suportes físicos que lhe sirvam de referência.

Estas características de interacção e comunicação das comunidades virtuais indiciam um ambiente adequado à aprendizagem e um potencial grande valor educacional pelo que a utilização de modelos de comunidades pode dar um importante contributo para o ensino.

Neste artigo procurar-se-á construir uma argumentação que suporte a utilização de comunidades virtuais ao serviço do ensino. A base da argumentação estrutura-se nas características sociológicas e tecnológicas das comunidades virtuais e nos modelos de aprendizagem construtivistas.

Na secção 2 descrevem-se as Comunidades Virtuais, na secção 3 as principais tecnologias relacionadas com Comunidade Virtuais, na secção 4 abordamos as comunidades virtuais ao serviço do ensino e, finalmente, na secção 5 as conclusões e perspectivas de trabalho futuro.

2 Comunidades virtuais

As comunidades virtuais vão-se construindo, progressivamente, com base num tráfego contínuo de mensagens. Este tráfego comunicacional constituído por um inúmero fluxo de informações e mensagens partilhadas vai tomando corpo e criando sua própria especificidade. Desta interacção e comunicação, representada na Figura 1, emergem então os primeiros contornos da identidade da comunidade.

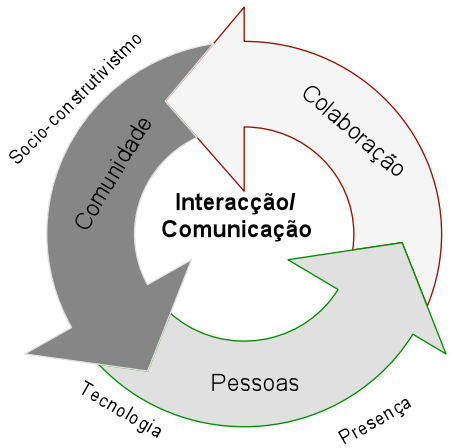


Figura 1 – Interacção e comunicação em comunidades virtuais [Adaptado de (Palloff, 2005)]

O fluxo de informações que se apresenta, ao mesmo tempo, como sendo fomentador de actividades e discussões sobre tópicos e assuntos diversos assume-se também, a partir de determinada altura como sendo um suporte para o estabelecimento de vínculos sociais incorpóreos entre os participantes. A sociabilidade que emerge a partir destas circunstâncias desenvolve-se de forma inédita, uma vez que não há registos na história humana de formas de práticas sociais de socialização com estas características.

As denominadas comunidades virtuais tiveram as suas origens remotas a partir de iniciativas localizadas em diversos pontos do mundo que nasceram, enquanto aglomerações de pessoas, de forma não coordenada e imprevisível.

Os primeiros promotores e arquitectos dessas aglomerações não tinham por intenção declarada o estabelecimento de comunidades ou a realização de acções conjuntas e orientadas para um determinado fim comum. Contudo, é possível detectar nesses pioneiros alguns pontos de convergência que motivaram a sua participação nesse primeiro momento do desenvolvimento das redes de comunicação por computador: o desejo de experimentar novas possibilidades promovidas pela tecnologia disponível no momento, a capacidade de improvisação e paixão pela criação de novos programas e consequentemente de novas utilidades para a tecnologia informática. Neste período inicial, o desenvolvimento tecnológico foi profundamente acelerado pelas pesquisas militares, e em particular pelas subsidiadas pelo Departamento de Defesa do Governo norte-americano, que visavam a construção de uma rede de comunicações que não fosse controlada centralmente, a fim de poder sobreviver a um possível ataque nuclear, temido durante o período da chamada Guerra Fria. Foi então criada a ARPANET (Advanced Research Projects Agency Net), a rede inicial de computadores que esteve na base da actual rede Internet (Castells, 2004).

Em 1983 a ARPANET separou-se em duas redes: uma rede de investigação que manteve o nome ARPANET e uma rede militar de nome MILNET. No final dos anos 80 a NSF (National Science Foundation) iniciou o projecto NSFNET para ligar diversos centros de computadores e respectivas comunidades de utilizadores. A rede NSFNET ficou activa em 1986 e tornou-se o principal suporte da Internet. O conjunto de redes interligadas que usavam TCP/IP (Transmission Control Protocol/Internet Protocol) e a ligação à NSFNET cresceu tanto, de forma espontânea e sem obedecer a qualquer plano central prévio (Rosa, 2003), que não era necessária a ARPANET como coluna vertebral do sistema. Função que seria mais tarde assumida pelas redes comerciais. Em Março de 1990 a ARPANET viria a ser desactivada. A possibilidade de ligação e de entrelaçamento de computadores situados em lugares geograficamente distantes e os mais diversos possíveis, através da utilização das redes telemáticas, permitiu a transmissão de dados ao nível global. A convergência destes avanços tecnológicos com os aspectos sociais, a possibilidade de estabelecer uma "nova" sociedade internacional com novos espaços de partilha e relacionamento entre indivíduos oriundos de diferentes povos foi desde logo adoptada e explorada pelos jovens entusiastas e apreciadores desse novo campo de exploração. Este "espírito comunitário" foi uma das principais causas do estabelecimento de um ideal que atravessou transversalmente todo esse momento de eclosão dos diversos agrupamentos sociais na comunicação em redes de computador.

3 Tecnologias

Para facilitar a criação das comunidades virtuais surgiram na Internet vários tipos de software específico com foco em diferentes áreas. Entre as diferentes áreas, podemos destacar o foco no entretenimento, na agregação e distribuição de notícias, na associação de pessoas em torno de temas ou interesses comuns, na colaboração académica e científica, na colaboração entre empresas e, ainda, o foco no ensino e aprendizagem.

O software utilizado para o desenvolvimento de comunidades virtuais centradas no ensino e aprendizagem traz consigo discussões sobre as pedagogias para o desenvolvimento de metodologias educacionais na World Wide Web. Os LMS (*Learning Management System*) são *softwares* dedicados essencialmente à gestão dos alunos e das actividades de aprendizagem, com recolha de dados relativos ao progresso dos alunos ao longo de um curso à distância na Internet. São exemplo de LMS, entre outros, o Saba e o Sum Total. Os CMS (*Course Management System*) são *softwares* que permitem a gestão do percurso do aluno, acompanhar e monitorizar o seu desempenho, criar e distribuir conteúdos, organizar e gerir actividades, avaliações, testes, disponibilizar ferramentas para comunicação e interacção entre as pessoas. São exemplos de CMS, o Moodle, o WebCT, o Blackboard, que entre outros, ganham espaço no quotidiano dos educadores virtuais pelo facto de possibilitarem uma fácil utilização e controlo de aulas, discussões, apresentações e, de uma forma geral, de todas actividades educativas virtuais.

O termo Web 2.0 ou Social Web é utilizado para descrever a segunda geração da World Wide Web onde se verifica uma tendência para o reforço da troca de informações e colaboração entre utilizadores e serviços virtuais.
Sob o ponto de vista do ensino e aprendizagem, a Web 2.0 aponta uma nova visão, na qual o aluno é capaz de encontrar nos diversos espaços do universo da Web 2.0 um conjunto de informações contraditórias dos conhecimentos obtidos nos processos de aprendizagem formais. Esta característica induz a discussão contínua dos factos, temas e assuntos tendo, por um lado a visão da existência de uma base comum de conhecimentos formais e, por outro, a reflexão conjunta em comunidade.

A Web 2.0 suporta-se em várias ferramentas e tecnologias base, das quais destacamos as seguintes:

- *Blogs*: abreviatura de *weblog*. Qualquer registo frequente de informações pode ser considerado um blog. A maioria das pessoas usa os blogs como diários pessoais, porém um blog pode ter qualquer tipo de conteúdo e ser utilizado para diversos fins. Uma das vantagens das ferramentas de blog é permitir que os utilizadores publiquem seu conteúdo sem conhecimento técnico especializado.
- *Mash-ups*: serviços criados pela combinação de diferentes aplicações que estão disponíveis na internet. Por exemplo, misturar um site de mapas on-line com um serviço de anúncios de locais turísticos para construir um mapa turístico único.
- *Wikis*: páginas na internet partilhadas por uma comunidade de utilizadores que podem ser alteradas por todos os utilizadores com direitos de acesso.
- RSS (*Really Simple Syndication*): forma de distribuir informação por meio da internet que é constituída por uma combinação de tecnologias "*pull*",

com as quais o utilizador solicita as informações que deseja, e tecnologias "*push*" , que permitem enviar automaticamente informações para um utilizador.

- Tagging: versão Web 2.0 das listas com sites preferidos, que oferece aos utilizadores uma maneira de associar palavras-chaves a palavras ou imagens que consideram interessantes na internet, ajudando, dessa forma a categorizá-las e a facilitar sua disponibilização a outros utilizadores.

4 Comunidades virtuais ao serviço do ensino

Sempre que se fala em inovação na educação, a palavra comunidade tornou-se obrigatória. Contudo, apesar desta evidência linguística, não é claro quais as características, pressupondo-se que existem algumas, que ligam os dois termos. Esta confusão é, sobretudo, comum nas comunidades virtuais onde, apenas fazendo um registo, escrevendo uma password ou pagando uma inscrição, qualquer pessoa que visite um Website se torna, automaticamente, um "membro"da comunidade. A ser verdade, isso significaria que um qualquer grupo de pessoas, agrupadas num espaço físico ou virtual, formariam automaticamente uma comunidade. Não se sabe muito sobre o valor educacional de utilizar modelos de comunidades para apoiar a aprendizagem. Para Hewitt (Hewitt, 2004, pág. 210) uma comunidade de conhecimento - KBC (*Knowledge Building Community*) - é um tipo de comunidade de prática caracterizada por: Partilha de conhecimento, valores e crenças; Pontos comuns de vivência entre os membros; Interdependência mútua; Mecanismos para reprodução; Práticas comuns; Oportunidades para interacções e participação; Relações pessoais significativas_; Respeito por diferentes perspectivas e por minorias

Se nas comunidades de investigação académica é fácil perceber que se espera que os seus membros, no seu dia a dia, realizem um trabalho contínuo na produção de novo conhecimento, por exemplo: escrever artigos, recolher dados e, fazer apresentações em conferências. No caso de se querer transformar uma sala de aula numa comunidade de conhecimento, isso obriga a uma mudança complexa nas regras da aula e dos papéis tradicionais dos professores e alunos. Para os estudantes o desafio já não é apenas completar as tarefas indicadas pelos professores, mas sim colaborar activamente na definição e resolução de problemas do seu interesse, desenvolver planos, aceitar desafios intelectuais, sintetizar ideias e trabalhar com os outros.

O desenho das comunidades de aprendizagem, apesar de ser algo distinto conforme o tipo de comunidade (ensino formal, desenvolvimento profissional, criação de conteúdos científicos) tem considerandos comuns. Segundo Cuthbert (Cuthbert, 2001, pág. 215), os considerandos comuns para desenhar com sucesso comunidades de aprendizagem são: Suportar as práticas actuais e as tarefas diárias dos participantes; Recolher experiências e representá-las de forma acessível e equilibrada; Disponibilizar uma *framework* para guiar o processo de aprendizagem; Representar as entidades dos membros da comunidade. Estas estratégias de desenho encorajam os membros da comunidade a partilharem as suas ideias, aceitarem as ideias dos outros e refinarem o seu próprio conhecimento, integrando o ponto de vista dos outros. Os

estudos sobre o desenvolvimento cognitivo humano relevam a importância dos contextos de aprendizagem em geral, e da importância do grupo e, essencialmente das trocas afectivas das relações interpessoais, como factores determinantes ao sucesso das aprendizagens.

Há muito que a célula familiar está identificada como a primeira comunidade de aprendizagem, onde as interdependências emocionais são o contexto das mudanças de desenvolvimento necessárias, para os mais jovens (Piaget) e para os Adultos (Baltes). As comunidades de aprendizagem para se assumirem como espaços de desenvolvimento pessoal e de transformação social necessitam de reorientar o enfoque da aprendizagem de competências específicos para o desenvolvimento da competência da aprendizagem (Fonseca, 1999). Para Vygotsky, sendo a aprendizagem um aspecto universal e necessário ao processo de desenvolvimento cultural, social e psicológico, a boa aprendizagem é aquela que está avançada em relação ao desenvolvimento. Vygotsky (Vygotsky, 1978) defendia que a aprendizagem originava processos internos de desenvolvimento que apenas operavam em interacção e em cooperação com outros. Uma vez assimilados, esses processos constituíam parte das aquisições do desenvolvimento independente do aluno. Porém, uma das principais preocupações de Vygotsky foi tentar perceber o mecanismo de desenvolvimento de processos mentais que, segundo ele, é mediado por ferramentas psicológicas como por exemplo a linguagem e o meio. O que sobressai em Vygotsky é o conceito de que as ferramentas psicológicas são sociais. Muitos são os modelos de aprendizagem identificados em Psicologia e os contributos da Psicologia da Educação ou para a Educação continuam a mostrar dificuldade em ultrapassar a Didáctica e invadirem as práticas reais dos actores educativos, evidenciando a velha discussão entre o "saber" e a interiorização desse saber, a aprendizagem e a sua utilização: O Conhecimento (Cerclé, et al, 1999). Os sistemas formais de aprendizagem continuam assentes em modelos de condicionamento operante (Skinner, Pavlov) e/ou interaccionistas (Piaget, Vygotsky, Bandura, ...) onde o professor / formador assume o papel de emissor do estímulo (o que tem para ensinar). A formação permanece, neste sentido, a ser uma relação de poder, desequilibrado, entre formando e formador, reforçando as experiências - maioritariamente negativas - vivenciadas ao longo do período escolar, induzindo respostas defensivas de quem se sente numa posição de fragilidade que, ainda que relativa é, amiúde, nesse contexto, vivida como geral, e dificultando a exposição do sujeito à abertura necessária para que ocorram mudanças cognitivas, as quais não se traduzem em apenas aprendizagens, mas à verdadeira produção do conhecimento, enquanto processo individual e único.

A aprendizagem para ter valor de transformação (desenvolvimento) cognitivo tem que ocorrer em contexto de mediatização. De acordo com Fonseca (Fonseca, 2000, pág.69) a: "Experiência de Aprendizagem Mediatizada (EAM) dá relevo ao mediatizador…, isto é, ao ser humano que se interpõe e intervém entre os estímulos e os próprios indivíduos mediatizados, com a intenção de mediatizar tais estímulos, adequando-os às suas necessidades específicas". Uma das dificuldades das modalidades de aprendizagem, descontextualizadas das interacções humanas, designadamente o e-learning, ou nas suas formas primordiais, os programas de aprendizagem por correspondência, resulta essencialmente dessa ausência de elemento mediador de aprendizagem, na base do qual o material de aprendizagem ganha valor suficientemente significante - ultrapassam o mero sentido do estímulo -

para serem interiorizados e, por essa via, terem real valor de evolução e de desenvolvimento.

5 Conclusões

Um dos maiores desafios que se coloca aos profissionais que actuam na área da Educação é o de criarem as condições necessárias para que os contextos educacionais sejam espaços onde os indivíduos consigam desenvolver a autoconfiança necessária para desempenharem o papel do aprendiz, mesmo quando "mestres", porque conscientes da relatividade do "seu" saber. As Comunidades de Aprendizagem são excelentes contextos para este exercício, permitindo a partilha de conhecimentos e o assumir alternado dos papéis de "aprendiz" e "mestre".

O saber enciclopédico não só deixou de ter aplicabilidade como se tornou uma impossibilidade: a velocidade da produção do conhecimento ultrapassou em larga medida a capacidade da sua armazenagem. A sociedade contemporânea acarreta o desafio da integração dos contraditórios: o "Saber" altamente especializado das diferentes ciências e a ineficácia desse mesmo saber senão aplicado de forma intra, inter e transdisciplinarmente. As comunidades de aprendizagem, pela heterogeneidade dos seus membros prometem ser particularmente úteis na promoção da transdisciplinaridade.

As tecnologias de informação actualmente disponíveis permitem a concretização destas comunidades de aprendizagem e têm vindo a evoluir no sentido de uma cada vez maior partilha e interacção entre os seus membros. Exemplo disto é a denominada Web 2.0 ou Social Web.

6 Referências

Castells, M. A Galáxia Internet: Reflexões sobre Internet, Negócios e Sociedade Fundação Calouste Gulbenkian, Lisboa, 2004.

Cerclé, A., and Somat, A. Manual de Psicologia Social. Epigénese, Desenvolvimento e Psicologia Instituto Piaget, Lisboa, 1999.

Cuthbert, A., Clark, D., and Linn, M. "WISE Learning Communities," in: Building Virtual Communities, K.A.S. Renninger, Wesley (ed.), Cambridge University Press 2001, p. 211.

Fonseca, V. Aprender a aprender, a educabilidade Cognitiva, (2.ª Edição ed.) Editorial Notícias Lisboa, 1999.

Hewitt, J. " An Exploration of Community in a Knowledge Forum Classroom," in: Designing for Virtual Communities in the service of learning, R.K. Sasha A. Barab, James H. Gray (ed.), Cambridge University Press, 2004, p. 451.

Lévy, P. Cibercultura ed. 34, São Paulo, 1999.

Palacios, M. "O Medo do Vazio: Comunicação, Sociabilidade e Novas Tribos," in: Idade Mídia, A.A.C.R. (Org.) (ed.), Editora da Universidade Federal da Bahia, Salvador, 1995.

Palloff, R.M., and Pratt, K. Collaboration Online - Learning Together in Community Wiley, 2005.
Rheingold, H. A Comunidade Virtual Editora Gradiva, Lisboa, 1996.
Rosa, A.M. Internet - Uma História, (2ª ed.), Lisboa, 2003.
Vygotsky, L.S. Mind in society, the development of higher psychological processes Harvard University Press, Cambridge, 1978.

Engenharia Conduzida por Modelos

Use of MS DSL Tools in the development process of a Domain-Specific Language

André Rosa[1], Vasco Amaral[1*] and Bruno Barroca[1*],

[1] Research Center for Informatics and Information Technologies,
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa,
Quinta da Torre, 2829-516 Caparica, Portugal
andredfr@gmail.com, vasco.amaral@di.fct.unl.pt, mailbrunob@gmail.com

Abstract. Given the popularity of Domain-Specific Modeling (DSM), metamodeling language workbenches have emerged to help the language designers. In this paper we analyze a specific tool, MS DSL Tools, and its adequacy to the development process of a diagrammatic Domain-Specific Language for the domain of Augmented Reality. We report some of the main challenges and suggest opportunities for improvements. Finally, we collect the more important statements that are valid and pertinent for the evolution of metamodeling language workbenches.

Keywords: Domain-Specific Language, Metamodeling Language Workbench, Development Methodology, Model Driven Engineering, Language Interaction Metamodel.

1 Introduction

Given the popularity of Domain-Specific Modeling (DSM), metamodeling language workbenches (MLWs) have emerged to help the language designers. This paper refers to the development process of a Domain-Specific Language (DSL) using a MLW. MLWs are used to simplify the development process of DSLs, but are still not mature enough. Software language engineers use MLWs to design their DSLs and rapidly prototype their respective graphic editors to be used by the domain experts.

On this paper, we describe the development process of a diagrammatic DSL using the MLW Microsoft DSL Tools [1] as the case study. DSL Tools is, together with MetaEdit+ from MetaCase [2] and EMF/GMF, Eclipse Modeling Framework [3] / Graphical Modeling Framework [4], one of the main MLW used in the industry. And we find it to be a good representative of the current state of the art in MLWs.

This DSL targets the domain of Augmented Reality, described in [5] and [6]. It is a medium complexity language that had a demanding and complex implementation over the MLW. For these reasons, its development process is well suited to evaluate the abilities of the used MLW.

* The results are partially supported by FCT- MCTES project PTDC/EIA/65798/2006.

The objective of this work is to analyze a specific MLW and evaluate its adequacy to the development process of a DSL. We highlight some of the main challenges and flaws, and suggest opportunities for improvements. It is not the focus of this work to detail all the features of the MLW, or to describe the DSL that was created using it. Also there are already some papers where the features of MLWs (e.g. exchange formats, model transformation support, specification of metamodel syntax and semantics), including MS DSL Tools, are analyzed [7]. However, none of them mention the usability and interaction problems we want to highlight. In spite of this work focusing only on one DSL and a specific MLW, we find it relevant and pertinent for the evolution of MLWs. The rest of this paper is organized as follows: on section 2, the background of this work is presented. Section 3 describes the analysis of DSL development using the MLW from the point of view of the software language designer. The domain experts usability report is shown on section 4. Finally on section 5, conclusions and future work are presented.

2 Background

2.1 Domain-Specific Language

A Domain-Specific Language (DSL) is a language that sacrifices generality for proximity to the problem domain. It is focused only on expressing problems from a specific domain, by using concepts from that domain to express problems at a higher level of abstraction. The conceptual and semantic distance between the problem space and the language used is reduced. Because the concepts presented on the DSL are from the target domain and not generic computer terms from the solution domain [8].

The main feature of any DSL is its expressiveness on the target domain. A DSL explores the regularities and standards of the domain. Using a DSL provides simpler, easier and more reliable programming. It also reduces the amount and complexity of development artifacts produced which translates into a better productivity and lower maintenance costs. They allow validation and optimization at the domain level, conservation and reuse of domain knowledge [8]. In some cases, DSL are considered end-user languages because they are supposed to be used by non programmers.

A DSL is normally declarative and provides a smaller set of notations and abstractions when compared to a general purpose language (GPL). DSLs are also known as micro language, little language, application-oriented language or special-purpose language [8]. These definitions, micro and little, refer to the tiny scope of the language and not to the number of concepts present on the DSL which is usually large. Some well known examples of DSLs are SQL, HTML, BNF, LATEX, GraphViz and the shell scripts of the Unix systems.

2.2 DSL Development Methodology

The DSL development methodology follows three main phases: domain analysis, modeling and implementation, as shown in Fig.1. On the domain analysis the problem domain is identified. Relevant domain knowledge is collected and transformed into

concepts and operations that will be used on the design of the DSL. The main sources of domain knowledge, as mentioned in [9], are technical literature, existing implementations, customer surveys, expert advice, and current and future requirements. The outcome of the domain analysis is the domain model.

On the modeling phase the language is designed. The design process of a DSL includes the definition of the abstract concepts (language's abstract syntax) used in the target domain. Then the representation of each abstract concept is mapped using a particular editor (language's concrete syntax). The concrete syntax represents the way of how the abstract concepts are to be symbolically presented to the domain experts. Generally, the concrete syntax is categorized into textual or visual/diagrammatic according to the presentation paradigms present on today's modeling frameworks. Finally, each abstract concept (or patterns) is mapped with its meaning (language's semantics) on a computational framework.

On the implementation phase it is built a custom editor for the DSL users to work with, and an automatic artifact generator. A software framework can also provide the base for a DSL implementation.

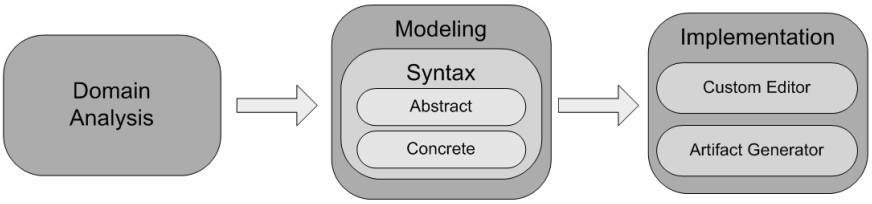


Fig. 1. DSL Development Methodology.

A fourth phase can be considered for when the DSL is deployed for the users to start to use it. The evaluation of the DSL usability may trigger again a new DSL development cycle. The automatic editor generation of the MLW usually become useless when most of the hand-code customizations break the automatic prototyping assumptions admitted by the MLW generation engine.

The creation process of a DSL is ruled by the same base principles that those of a GPL: helps the user, to have a simple syntax and semantic, to achieve a good level of abstraction and expressiveness capacities, and to be orthogonal. In the case of visual DSLs, special care should be taken with the usability of the editor.

2.3 Implementing DSLs with Model Driven Engineering

In Model Driven Engineering (MDE), models are the main development and maintenance artifacts of an application [10]. A model is used to construct abstractions that define the properties and features of a system [11].

Models are first class entities allowing a better comprehension of the domain problem concepts, and ready to run code is automatically generated from them. The usage of models as development entities, bring advantages like: a cleaner architecture presentation, conceptual simplicity, efficient implementations, scalability and

flexibility [11]. In MDE, a system is represented by a model that is in conformity with a metamodel. To achieve the full potential of MDE it is necessary that the modeling concepts refer to domain concepts instead of being mapped into generic technological concepts [12].

2.4 MS DSL Tools

Using a MLW simplifies the development process of a DSL. It supplies all the basic necessary functionalities that are needed to build a DSL solution. It enables and guide the language designer to build the language metamodel, and the elements representation, to setup the rules and validations to be run on the models, to create the graphic editor to be used by the DSL users, and to specify the code generation process.

As mentioned there are other MLWs like MetaEdit+ and EMF/GMF. This one, Microsoft DSL Tools [1], was selected because of its implementation language. Nevertheless all of these MLWs are comparable and there is no difference regarding the usability aspects. The decision factor to use DSL Tools (using the latest version available Visual Studio 2008 SDK 1.1), was that it shared the same implementation language, C#, as the target framework. Which allows an easier code transformation from the DSL model to the framework code.

One of the key features of this tool is a set of Application Programming Interfaces (APIs) called the In-Memory Store, that is automatically specified during the DSL creation process. It provides a set of facilities to support the behavior of a DSL including creation, manipulation and deletion of model elements, links and access to the domain model. The base concepts provided by this tool to build the metamodels are domain classes and relationships. For the presentation connectors and a vast range of visual elements are provided, like shapes, compartments and swim lanes. The models are saved using Extensible Markup Language (XML).

Constraints
DSL Tools uses C# code to define constraints. Constraints are used to extend the correctness checking of the models beyond the abstract and concrete syntax of their modeling language. They allow the specification of complex interrelationships between the model data. Soft constraints are wrapped into a validation method that defines its structure and error/warning messages.

Generation methods for artifact generation
DSL Tools provides three techniques that can be used to generate artifacts [1]: Extensible StyleSheet Language Transformations (XSLT); using a domain-specific API; and using a template based approach. Any of these techniques can convert the XML model built on the editor into implementation artifacts such as C# classes or XML files. The use of XSLT usually results in extremely quick model-to-code transformations. But has the downside of the XSLT being non trivial to create and to understand as there is a lot of string manipulation occurring. The second technique uses the API provided by the In-Memory Store and standard .NET code. Still the code to be generated is hard-coded with individual strings that reduce its legibility. Finally,

the third technique is based on text templates. It uses a specific derived template base class that provides access to the language model In-Memory Store. There is a clear separation of concepts between the elements that are rendered directly on the output file, and the elements that are used to provide structure and dynamic behavior of the generation process. This is achieved using special characters for delimiting the text commands. This can be a gradual process dealing with a specific type of model element at a time. The readability of a text template is still not optimal but from the three techniques, this is the closest to the desired output.

DSL Tools generated editor

The main common elements in a DSL Tools generated editor, as seen in Fig.2, are the toolbox, design surface, model explorer, solution explorer, messages window and properties window. The toolbox is where the language elements are placed and divided into categories. The design surface is where the language elements are placed (standard zoom-in and zoom-out features are provided). The solution explorer presents a tree like representation of the files that belongs to the solution. The model explorer displays a tree like presentation of the model elements. The properties window displays the parameters of the current selected model element. Finally, in the messages window, error and warning messages are presented. It is also on this window that the language validation messages are displayed.

As the editor provides all these options from the start, some users may have difficulty identifying where to start and what to do. The generated editor should be customizable to the needs of each domain expert profile, offering different abstract edition models. For less experienced users, the use of a wizard to help and guide would simplify and decrease their learning-time of the editor.

The default common interaction method of the generated editor is based on drag-and-drop. The user selects from the toolbox the element to add to the model, and places it on the correct place on the design surface. It is also possible to write code and define more advanced forms and context menus for each shape.

One important note about the DSL Tools is that it provides a visual editor with a base set of features and functionalities. It is possible to go beyond those base functionalities by constructing new functionalities using code. For even more advanced functionalities it is required an understanding of the Visual Studio internal architecture that hosts the DSL Tools. This is a reason why this solution does not scale so well. For more demanding customers it is necessary to create non trivial code that will become increasingly harder to understand. Therefore, the prototyping capacities of this MLW starts to fade away. As it is essential to identify the points where code will need to be introduced and the specification of the code to place there.

2.5 Augmented Reality Language

In our case study we developed a DSL called Augmented Reality Language (ARL), which is dedicated to the particular domain of specifications of Augmented Reality Software applications. The ARL is used to define the application logic's organization using a workflow-like presentation schema, while providing means to configure it. It is also used to define the virtual reality world objects and their behavior. A full

description of this language can be found in [5]. Implementation details and user evaluation results were presented in [6].

It focus on improving the productivity during the software development process of Augmented Reality applications. This DSL was built using a combination of meta modeling theory, visual languages and DSLs. A metamodeling language workbench (MS DSL Tools) was used to define the language's concrete and abstract syntax. Its semantics interpretation was achieved by generating code from our models to a target framework. The evaluation process [6] used a series of tests, and a case study where domain experts validated the DSL's efficiency, effectiveness and expressiveness. Fig.2 presents the ARL DSL Tools editor.

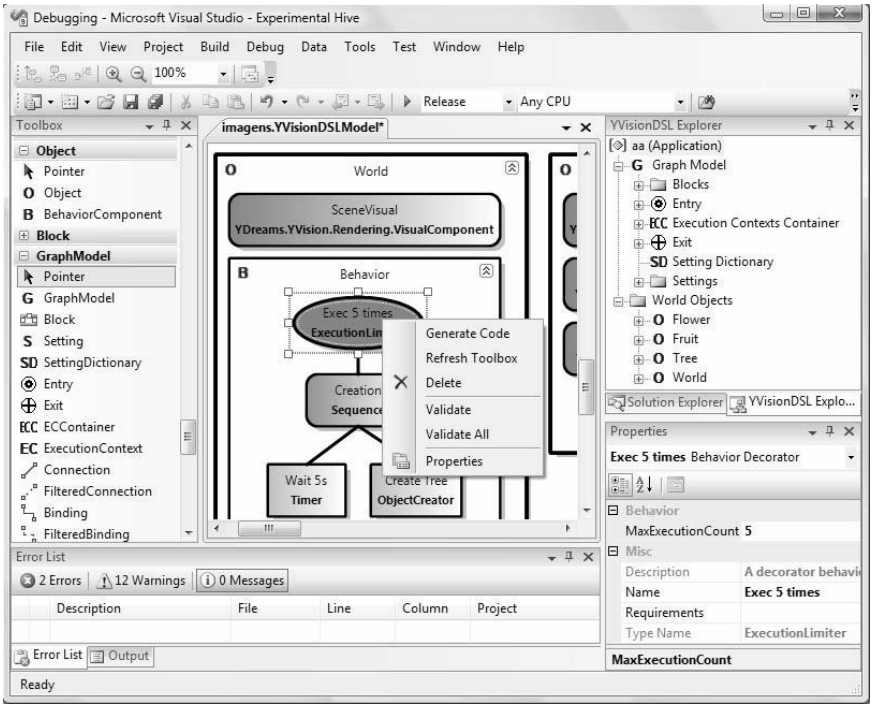


Fig. 2. ARL DSL Tools editor.

The Augmented Reality Language DSL is a complex language and its development process followed all the phases of the creation process of a DSL. This includes the creation of a complete editor and code generation to a real C# framework (YVision) from YDreams[1], and an analysis of the users feedback about the use of the DSL. Furthermore, this is not a purely academic exercise, but a real and concrete medium

1 This DSL is the outcome of a collaboration project between the Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (DI/FCT/UNL) with the Portuguese software house YDreams [13].

level complexity DSL, built to be applied on the software development industry context. This is why it makes a good example for the evaluation of the MLW which was used to implement it.

3 Analysis of DSL Development

3.1 Domain Analysis

On the domain analysis phase, the main language requirements and domain rules were enumerated, and most of these rules became constraints later on. In spite of this being one of the most important phase of the DSL development, the MLW provided no special support for this phase. There is no direct connection between the language requirements specification and the MLW implementation.

3.2 Language Design

The creation process of languages is still in consolidation, it has not reached the point of full maturity. Therefore, the language's design phase can be classified as an "artistic process" that starts when the domain analysis ends, and its outcome is obtaining a language specification. In spite of the existence of several different and configurable parts, it was decided that the DSL (as a whole) would have a consistent and unifying organization and presentation. The usage of nodes and connections was adopted as the base for the language design. The used MLW, like the others (MetaEdit+ and EMF/GMF), provide no particular support for this phase of the DSL development. Some kind of customizable templates for language design (reusable patterns) could have been useful in order to guide and help the software language engineer.

3.3 Language Implementation

The usual phases of a DSL implementation in a MLW are the definition of the language abstract syntax and concrete syntax. Normally, it is also necessary to specify additional constraints, and details about the graphical editor that will be provided to the DSL users.

Abstract Syntax
The language abstract syntax is defined using a metamodel. Each MLW has its own particularities on the definition of the metamodel. Therefore, as it was the case, it is sometimes necessary to slightly adapt the original DSL metamodel so it can be implemented using a particular MLW. The implemented metamodel is very complex. It has 18 main elements and several secondary elements. As such, it turned out to be difficult to visualize specific elements and their relationships. This is the result of an organization issue of the MLW. Proving that the MLW does not scale so well with a large language. A mechanism to compose the language metamodel, using packages

(like EMF/GMF uses), or another structure as a way to organize the metamodel should be provided. Also a system of user-defined views (7 in our language) would help the language engineer to cope with the visual complexity of the metamodel specification. All this would allow an easier abstract language validation process with the domain experts.

Concrete Syntax

The visual representation of the DSL concepts follows the logic of nodes and connections, where the nodes are placed inside the boundaries of other nodes.

The concrete syntax is defined by mapping each language concept to its representation. This is a simple and quick process to execute, but has the side effect of being too tied to the metamodel definition. This is especially true in the case of container elements that are represented as aggregations in the metamodel. Furthermore, the container concept is one that is not part of the standard meta-metamodel definition of DSL Tools, so it is a responsibility of the language engineer to construct it. In spite of providing various connectors and a vast range of visual elements like shapes and swim lanes, more flexible options could have been provided. In fact, it feels like the language representation is bound by the restrictions of the options provided by the MLW. Regarding the DSL editor, its layout is based on the same layout the language engineer uses. This is in most cases good enough, but it should be possible for the language engineer to build a editor from scratch using predefined components (e.g. toolboxes, buttons, message window). Furthermore, it is not possible to define or use other kind of interaction models (wizards, 3D editors, textual-diagrammatic mixed editors, etc).

Constraints

Code was used to define the constraints and also to specify when to check them. This solution allows for the definition of very complex constraints but does not have the simplicity of a solution like OCL (Object Constraint Language) present in EMF/GMF. A better mechanism to define constraints and specify their verification points should be provided. A solution could be the use of a new visual language built upon the DSL to define its constraints. This would allow the language engineer to build constraints using a graphic representation within the domain terms.

Given the cases where the same language is used by users with different profiles and access permissions, it should be possible to define different access levels. Each of these access levels would have less or more restrictive constraints. These more restrictive constraints could provide guidance for less experienced domain experts.

Another important aspect of constraints is their feedback to the users: when a constraint is violated, a message and/or visual change is passed to the user. This MLW does not make a distinction between actions resulting from constraint violation from normal edition actions, nor it provides a structured way to define the visual feedback. To do this kind of reactive actions, like color or form change when the introduction of an invalid value, it is again necessary to write code.

Generic Elements

One of the features of the ARL is the use of generic elements [5]. Each generic element defines an interface, and those interfaces correspond to the main building

blocks of the language. Basically, a loaded concrete element exists as an attribute of the generic element, and the generic element supplies a communication interface to it. This allows new building blocks to be created and used on the DSL according to their interface. Again, to implement this feature it was necessary to go beyond the normal use of the MLW. It required the use of code, and a lot of "digging" in the DSL Tools API and organization, to make it happen. This is another good example of a specific problem where the software language engineer needed to go beyond the normal use of the MLW and "dig" into its internal code. This situation is unavoidable, but it could be made easier if the MLWs have that in consideration, and also present their internal organization specified as a model.

Editor complementary features

The default implementation of the editor already solved some challenges like providing a quick way to edit an element's name, automatic routing of the connections between language elements, and zoom in/out of the design surface.

Still, it was necessary to implement a set of visual functionalities whose sole purpose is to enhance the user experience. They add no new features to the language definition but are especially useful to manage the scalability of the models and to overcome the shortcoming normally associated to visual programming languages. The types of enhancement they provide are: center view on an element, hide/show of unused elements, instant visual feedback of model changes, and presentation of additional information in selected places. These features are not part of the initial DSL definition, but are important for the user experience when using the editor. We had no way to express these features on the MLW except using code, and they are part of the interaction model of the editor.

3.4 Language Semantic

The semantic of the DSL was defined by generating code from the models to the target framework. To do that, we used text based templates. They ended up being a good solution as they make a separation of concepts between the elements that are rendered directly on the output file, and the elements that are used to provide structure and dynamic behavior to the generation process.

In regards to the organization of the generation process, it was lacking a good distinction between implementation independent code generation and dependent code generation. In most situations, the code to be generated depends on the used implementation technology but the meaning of the generated code remains the same, it is only represented in a different way. Therefore, to use another target framework in the generation process it will be necessary to rewrite almost all of the generation process. This happens because the MLW does not provide a way to express the code generation mechanism. One of the requirements for the code generation was the generation of organized model-related code. This means that the identifiers used on the generated code are the same that are present on the model. This allows the use of a traceability framework to identify which part of the model caused the runtime error tracked to a specific line of generated code. This also has the side effect of allowing the generated code to be extended and customizable by developers. To do this, it was

necessary to do more work on the generation rules present on the text templates and to add additional verifications (e.g. valid C# identifier, prevent collisions between identifiers) to the models to ensure correct and valid identifiers for the elements to be generated. These verifications ended up being on the same level (of abstraction) as the model constraints, since it was not possible to distinguish between domain restrictions and code generation related restrictions.

4 Usability report

Fig.3 presents an overview of the DSL solution that was created.

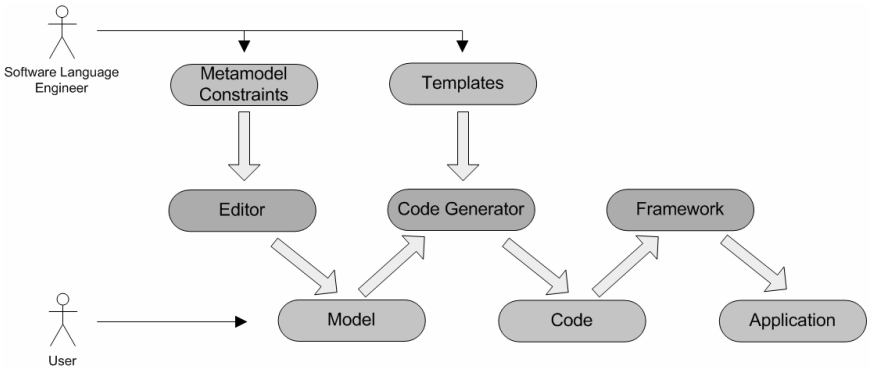


Fig. 3. DSL solution overview.

There are two actors: the software language engineer and the domain expert user. The software language engineer defines the metamodel and constraints used by the editor and the templates used for the code generator.

The editor, code generator and framework are methodology actuators. The model, code and application are artifacts. The user uses the front end (the editor) to help him express his mental model on a new DSL model. The user only contacts with the model representation given by the editor, and with the application itself. The implementation level details are supposed to be hidden to the user. All the transitions represented by the larger arrows are made without human intervention.

User comments

In spite of the measurable gains in productivity [6] there are still some issues that could be improved. From the evaluation process that was conducted to YDreams's Augmented Reality domain experts, a set of usability problems were identified [6]. The interface details that were suggested to be enhanced are: option to keep a connector tool selected after a connection is made, and when making connections the drop space should be bigger. These details are related to the default generated editor functionalities that are not matured enough to consider this kind of advanced user support. The most important extra functionalities mentioned by the users were

enabling copy / paste of the elements, and ability to select multiple elements of the same type and configure them all at once. Also mentioned by the users was the support for optional forced placement of elements in the design surface. These functionalities make sense, and are important in providing a better productivity for the users. Therefore, they should be standard in every MLW. In DSL Tools, that is not the case, as the implementation of these functionalities requires some profound changes and customizations that need to be made by the language engineer.

Furthermore, the use of the toolbox entries is overwhelming. A solution to this was the creation of element specific context menus. Context menus are not part of the base functionalities provided by the MLW and had to be created using non trivial code.

Interaction Model

The problems detected by the users are not related to our language definition. They are related to the MLW. Some of the minor problems related to Human-Computer Interaction (HCI) are easy to identify but hard to solve using the MLW. The existence of users with different profiles (such as programmers and designers) also makes it harder to define a single interaction schema. The best approach would be to provide different interaction schemas all based on the same interaction model.

When dealing with productivity, a very usable interaction model is required. Even if the DSL has the right concepts with the right representation, it is necessary that it also provides an interaction model for the users. The interaction model encapsulates the DSL syntax (abstract and concrete), and the users interact with the DSL through it. Fig.4 represents this interaction.

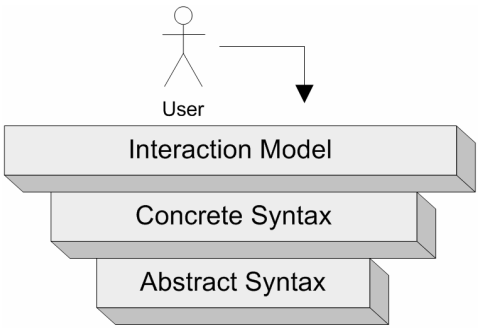


Fig. 4. DSL Interaction Model.

The language designer usually can dedicate more time to learning the language workbench functionalities. But his work could also benefit from a better interaction model. With a proper interaction model definition, it would be possible to automatically measure productivity using that same model (e.g. according to HCI usability guidelines). Therefore justifications for the users productivity while using a specific interface would be directly linked to the quality of the Interaction Model. It would not be necessary to conduct as many usability tests, speeding up the language development process.

5 Conclusion and Future Work

The main contribution of this work is to highlight the typical interaction problems found while using a specific MLW, and for that we used a diagrammatic DSL for the domain of Augmented Reality as a case study. When implementing DSLs with MLW there are problems that are not part of the DSL definition but part of the implementation technology. Given the increased interest in DSLs the current MLW are reaching their expressiveness limits.

MLWs need to support the definition of the user interaction model, and are still not fully adequate to all the stages of the DSL development methodology. It is still necessary to use code to define important functionalities of the generated graphical editors, and doing that usually means that we lose some prototyping abilities provided by the MLWs.

This work will continue surveying DSL implementations using several MLWs (e.g. MS DSL Tools, MetaEdit+, EMF/GMF and the under development AToM3 [14]) in search of challenges and respective solutions to pave the way to the creation of a HCI-based model specification for the definition of MLWs.

Acknowledgments. A note of appreciation goes for the YDreams company for allowing the use of their framework as the target of the DSL.

References

1. Cook, S., Jones, G., Kent, S., Wills, A.C.: Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley (2007)
2. MetaEdit+, http://www.metacase.com/
3. Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/
4. Graphical Modeling Framework, http://www.eclipse.org/modeling/gmf/
5. Rosa, A., Amaral, V., Barroca, B.: Designing a DSL solution for the domain of Augmented Reality Software applications Specification. Springer's LNCS. Proceedings of the 4th International Conference on E-Learning and Games (2009)
6. Rosa, A.: Designing a DSL solution for the domain of Augmented Reality Software applications Specification. Msc. Thesis. Universidade Nova de Lisboa. Portugal (2009)
7. Saraiva, J., Silva, A.R.: Evaluation of MDE tools from a Metamodeling Perspective. Information Resources Management Association. Journal of Database Management, 19(4), 50-75 (2008)
8. Deursen, A., Klint, P., Visser, Joost.: Domain-Specific Languages: An Annotated Bibliography. SIGPLAN Notices. 35(6): 26-36 (2000)
9. Thibault, S.: DomainSpecific Languages: Conception, Implementation and Application. Phd. Thesis. University of Rennes. France (1998)
10. Kelly, S., Tolvanen, J.: Domain-Specific Modeling. Wiley (2008)
11. Bezivin, J.: On the Unification Power of Models. Software and System Modeling (SoSym) 4(2): 171-188 (2005)
12. Booch, G., Brown, A., Iyengar, S., Rumbaugh, J., Selic, B.: An MDA Manifesto. MDA Journal (2004)
13. YDreams, http://www.ydreams.com/
14. AToM3, http://atom3.cs.mcgill.ca/index_html

Reverse Engineering of GUI Models[1]

André M. P. Grilo[*], Ana C. R. Paiva[*], João Pascoal Faria[*,§]

[*] Departamento de Engenharia Informática, Faculdade de Engenharia da Universidade do Porto, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal
[§]INESC Porto, Campus da FEUP, Rua Dr. Roberto Frias, nº 378, 4200-465 Porto, Portugal
{andre.grilo,apaiva,jpf}@fe.up.pt

Abstract. Graphical user interfaces (GUIs) are an important part of today's software and their correct execution is a very important requirement in order to ensure the effective use of the overall application. One way to find defects in GUIs is to test them by executing test cases and verifying the execution outputs. These test cases can either be created manually or produced automatically from a model of the GUI. It is unpractical to do extensive manual testing because it's a very time-consuming process; however creating a model of the GUI in order to generate automatically test cases is a very difficult task. This paper presents an approach for diminishing the effort required for constructing the model of an existing GUI. The technique is dynamic and exercises the GUI extracting information about the structure of the GUI and some of its behavior. This model is used in the context of model-based GUI testing.

Resumo. As interfaces gráficas com o utilizador (GUI) são uma parte importante do software dos nossos dias e a sua correcta execução é preponderante para assegurar a legitimidade de toda a aplicação. Para encontrar defeitos nas GUIs é comum testá-las executando um conjunto de casos de teste e verificando os resultados produzidos. Os casos de teste podem ser criados manual ou automaticamente a partir do modelo. Não é praticável testar manual e exaustivamente as GUIs. Contudo, construir um modelo para gerar testes automaticamente é uma tarefa muito difícil. Este artigo apresenta uma abordagem para diminuir o esforço necessário para construir modelos de GUIs, a técnica usada é dinâmica e exercita a GUI extraindo a informação sobre a sua estrutura e o seu comportamento. Este modelo é usado no contexto de teste baseado em modelos.

Keywords: Reverse engineering; GUI modelling; GUI testing.

[1] Work supported by FCT (Portugal) under contract PTDC/EIA/66767/2006.

1 INTRODUCTION

Nowadays' software systems usually feature Graphical User Interfaces (GUIs). GUIs are the mediators between systems and users and their quality is a crucial point in the users' decision of using them. GUI testing is a critical activity aimed at finding defects in the GUI or in the overall application, and increasing the confidence in its correctness. However, GUI testing is a very time consuming Verification and Validation (V&V) activity. The application of model-based testing techniques and tools can be very helpful to systematize and automate GUI testing.

Still, the effort required to construct a detailed and precise enough model for testing purposes (in order to be able to generate not only test inputs but also expected outputs), together with mapping information between the model and the implementation (in order to be able to execute abstract test cases derived from the model on a concrete GUI), are obstacles to the wide adoption of these techniques.

One way to relief the effort mentioned is to produce a partial "as-is" model, together with mapping information, by an automated reverse engineering process. This model will have to be validated and detailed manually, in order to obtain a complete "should-be" model at the level of abstraction desired. Some defects in the application can be discovered in this stage. Overall, the goal is to automate the interactive exploratory process that is commonly followed by testers to obtain a model for an existing application.

This paper is divided in seven sections: the first one is an introduction to the subject of this paper; section 2 is a review of the different reverse engineering approaches and some existing projects about this subject; section 3, presents our approach and section 4 describes the algorithm of the reverse engineering process used; section 5 presents some behaviors that are possible to identify in a GUI, and the necessary rules to infer them; section 6 presents the results generated by our approach applied to the Microsoft Notepad software application; the last section presents some conclusions and future work.

2 STATE OF THE ART

Information systems are critical to the operations of most businesses, and many of these systems have been maintained over an extended period of time, sometimes twenty years or more. Nowadays, many organizations are choosing to reengineer their critical applications to better fit their needs and to take advantage of the new technologies.

Reverse engineering is defined by Chikofsky and Cross [1] as the process of analyzing a subject system in order to identify the systems components, their relationships and to create representations of the system in another form or at a higher level of abstraction. Reverse engineering normally involves extracting the design artifacts and building or synthesizing abstractions that are less implementation-dependent.

For fully understanding existing software, it is necessary to extract both static and dynamic information. Static information embraces usually software artifacts and their relations [2]. Examples of artifacts are classes, methods, and variables. The relations could embrace extending relationship between classes or interfaces, method calls between methods, containment relationships between classes and methods or variables etc., information that can be retrieved from the analysis of the source code. On the other hand, dynamic information not only includes software artifacts, but also contains sequential information, information about concurrency, code coverage, etc.

There are two approaches for reverse engineering: a static approach, in which the static representations of the system (source code) are analyzed without executing the system[3]; and a dynamic approach, in which the system is executed and its external behavior is analyzed [4][5]. The static approach requires access to the source code of the system, which is not always available. Static approaches are particularly well suited for extracting information about the internal structure of the system and dependencies among structural elements. Besides decompilation techniques, dynamic approaches are the only option when the source code is not available. They are well suited to extract the physical structure of the system GUI and some of its behavior, but are more difficult to automate. We focus on dynamic approaches because our goal is to extract information for black-box testing purposes.

There are several examples of projects that use both approaches, one of them is CelLEST [6]. CelLEST is composed of two middleware tools (recorder and pilot). The main purpose of this project was to facilitate the migration and optimization of the uses of a legacy system on a new platform. In order to collect the necessary information to complete the possible task on the old system, CelLEST constructs a map of the system interface and a task model which describes the screen transitions that the user has to traverse in order to accomplish a task.

Another project that uses both approaches is ReWeb [7], the main purpose is to avoid the inevitable degradation of web sites by restructuring and rewriting the site. First it extracts the structure of the website then it analyses the html pages in order to detected syntax errors and other problems and to tune up the model.

Existing automated dynamic approaches try to explore automatically the system through its GUI, but may get blocked because they are not able to find the proper values for accessing all the system user interface elements and exercising all its functionality. E.g., this may happen when a login/password is required to proceed. To overcome this problem, we propose a hybrid approach that combines automatic and manual steps.

3 OVERVIEW OF THE REVERSE ENGINEERING AND TESTING PROCESS

Our reverse engineering process main purpose is to diminish the effort required for constructing the model of an existing GUI for model based testing purposes.

The tool described in this paper is capable of building a preliminary model in Spec# - a pre/post specification language [8] - by interacting with the existing GUI.

The model obtained by the reverse engineering process captures structural information about the GUI (the hierarchical structure of windows and interactive controls within windows and their properties) and also some behavioral information. The model describes the state of each window and window control (enable/disable status, content of text boxes, etc.) and the actions the user can perform on the window controls (e.g., press a button, fill in a text box, etc.). Typically, the preliminary model obtained by this process needs to be completed manually with additional behavior, for instance, some executable method bodies cannot be extracted automatically by the tool.

The final model in Spec# goes through a validation process (to ensure that it describes the correct behavior of the GUI) and then it is used to generate a test suite automatically, using the Spec Explorer tool [9]. A test suite is a set of test segments with sequences of operations that model user actions (with input parameters) interleaved with operations to check the outcomes of those actions.

Test execution is also supported by the Spec Explorer tool. In order to do that, mapping information relating model actions with real actions over GUI controls is needed. The mapping information is gathered during the reverse engineering process and is saved into a XML file. This file keeps information about physical properties of the GUI controls (in order to identify the GUI controls during test case execution), and keeps information about the mapping between abstract and concrete actions. Real actions are written in C# and are capable of simulating user actions over real GUI controls. During test case execution, related actions run in both the specification and implementation levels, in a "lock-step" mode, and, after each step, the results obtained are compared. Whenever an inconsistency is detected, it is reported.

The process of reverse engineering is performed by a dynamic approach which executes the application under test; the structure and behavior of the GUI are extracted and identified. The algorithm of the reverse engineering process is explained in more detail in the sequel.

4 DESCRIPTION OF THE ALGORITHM

The reverse engineering process constructs a GUI model in two phases.

The first phase of the reverse engineering process aims to extract physical properties of the GUI controls and also the navigation map among the different windows of the GUI application under test. The algorithm is as follows:

Phase 1 - Gathering structural information, Fig.1:

1. The user starts the application and the reverse engineering tool, and points out the application starting window.
2. The reverse engineering tool extracts information (physical properties) about all the GUI controls inside that window, and records it in a XML file.
3. The user interacts with GUI controls inside that window in order to open another window of the same application. The reverse engineering tool saves all the steps

performed by the user to navigate from the source window to the destination window in the XML file (in order to be replayed in the second phase of algorithm for opening windows).

4. If a new window was opened go to step 2 until there are no more new windows or the application is closed.

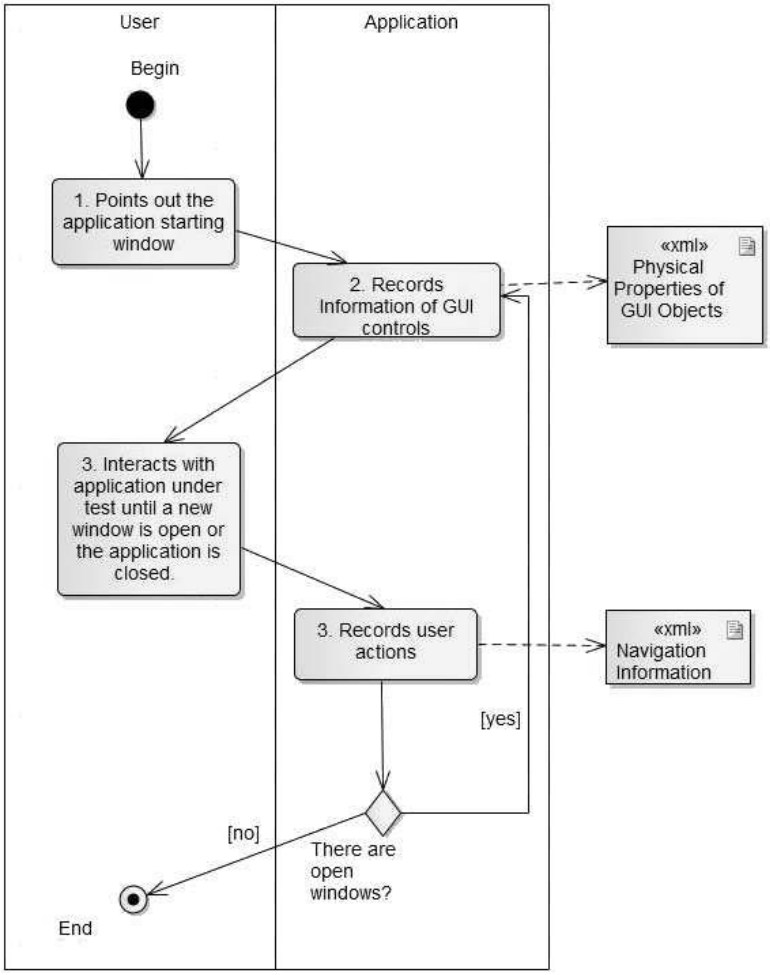


Fig.1. Gathering Structural Information

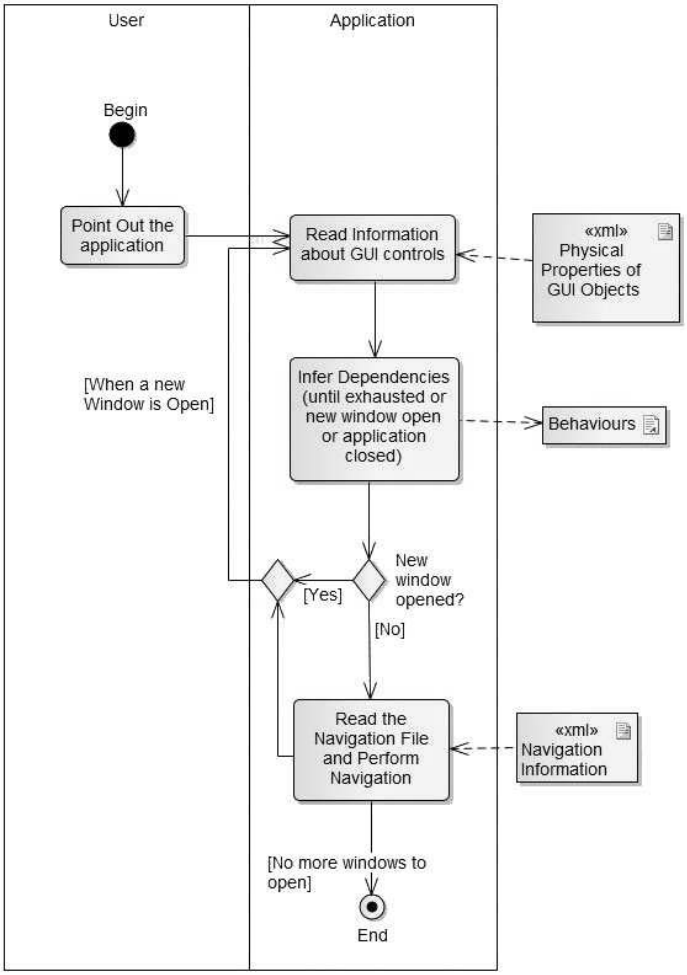


Fig.2. Gathering Behavioral Information

The second phase of the reverse engineering process aims to extract behavioral information, namely dependencies among GUI controls inside each window. The information gathered in the first phase is used here to navigate among windows. The algorithm is as follows:

Phase 2 - Gathering behavioral information, Fig.2:

1. The tester points out the starting window.
2. The tool reads information about GUI controls in this window from the XML file produced in phase 1.

3. To infer dependencies among GUI controls in the current window, the tool interacts with them and checks the changes produced on the properties of other GUI controls, until all controls and actions have been exercised. The dependencies discovered are saved in an XML file.
4. Based on the information captured in phase 1, the tool checks if there is any window that can be reached from the current one and has not yet been explored. If it is the case, the tool replays the steps recorded in the previous phase in order to navigate to that window and the algorithm proceeds to step 2. Otherwise, the exploration stops.

5 RULES TO INFER BEHAVIOUR

In order to identify the different kinds of behaviors that are common in GUIs, it is necessary to define some rules to infer the behavior. To discover the different behaviors three steps are performed: firstly, the actual state of the application is saved; then some action is applied to a GUI control; finally, the final state is compared with the initial state in order to infer dependencies among GUI controls.

Without restrictions, the state space S of an application comprising a set of GUI controls $O = \{o_1, ..., o_N\}$ will be the Cartesian product of the domain values of the GUI controls properties (p_i), i.e., $S = dom(o_1.p_1) \times dom(o_1.p_2) \times ... \times dom(o_1.p_m) \times ... \times dom(o_N.p_1) \times dom(o_N.p_2) \times ... \times dom(o_N.p_k)$[10]. There is a distinguished initial state s_0 that represents the initial state when the application is started. The set of properties of a GUI control o depends on its type and is denoted by $Properties(o.type)$. The value of a property p of a GUI control o in state s is denoted by $s.o.p$.

Depending on their type and state (enabled/disabled), each object accepts user actions (e.g., press a button, set text, etc.). Some of these actions may have parameters (e.g., set text). The set of all possible actions that can be performed on GUI controls is denoted by A. The set of actions available in a GUI control o is denoted by $Actions(o.type)$ and is a subset of A.

Performing an action a with parameters par on a GUI control o in a GUI state s may cause a transition to a new state s'. Each transition is described by the triggering user action (GUI control, action and parameters), source state and target state. The set of possible transitions is denoted by the transition function $T: A \times PAR \times O \times S \rightarrow S$.

In the case where actions need parameters, there is a configurable set of predefined values to explore, for example, the action `setText` may have $ParamValues(setText) = \{'a', 'A', '1'\}$.

An important dependency among GUI controls is the modification of a property p of a control o' when an action a is performed on another control o. The set of this kind of dependencies can be formalized by a set of tuples

$$M=\{<a,o,p,o'> | a:A,\ o:O,\ o':O,\ p \text{ is property of } o',\ o \neq o'\}.$$

The algorithm to extract these dependencies is as follows:

```
s0,s,s':S, a:A, o:O
M={}
s=s0
Forall o
  Forall a in Actions(o.type)
    if ParamValues(a) ≠ {}
       Forall par in ParamValues(a)
         s':= T(a,par,o,s)
      M := M ∪ {<a,o,p,o'>|o':O, p:Properties(o'.type)
           and s'.o'.p ≠ s.o'.p and o ≠ o'}
        s := s'
    else
      s':= T(a,null,o,s)
      M := M ∪ {<a,o,p,o'>|o':O, p:Properties(o'.type)
           and s'.o'.p ≠ s.o'.p and o ≠ o'}
      s := s'
```

The above algorithm can be specialized in order to find special kinds of dependencies:

- **Enabling/disabling dependency**. Actions over some GUI controls can enable or disable other GUI controls in the same window. E.g., in the "Find" dialog of the "Notepad" application, the button "Find Next" is only enabled when text is inserted in the textbox "Find What" (see case study).
- **Value propagation dependency**. The contents of GUI controls may depend on the values of other GUI controls (e.g., summation). To find these dependencies, the algorithm will explore only the action "update text" on GUI controls of type "textbox" (`O'= {o | o.type = textbox}`).
- **Master detail dependency**. An example of this behavior is when updating a Combo Box or List box causes changes to the content of a grid or list view.

As soon as the exploration process is finished and the XML file is complete, we generate a Spec# model by applying transformations to the XML file.

The reverse engineering algorithm was implemented on top of UI Automation [11] which is the new accessibility framework for Microsoft Windows, available on all operating systems that support Windows Presentation Foundation.

6 CASE STUDY

The case study was based on the Notepad application. The reverse engineering tool extracted the structure of the Notepad and recorded that structure into an XML file. This file saves information about windows (<window>), the navigation steps (<steps>), controls (<control>) and dependencies among GUI controls (<dependency>). An excerpt of that file is shown below:

```
<?xml version="1.0" encoding="utf-8" ?>
<applicationxmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <window>
    <name>Find</name>
    <AutomationID />
    <ControlType>Dialog</ControlType>
    <L_Controls>
    ...
      <Control>
        <name>Find Next</name>
        <AutomationID>1</AutomationID>
        <ControlType>button</ControlType>
      </Control>
      <Control>
        <name>Find what:</name>
        <AutomationID>1152</AutomationID>
        <ControlType>edit</ControlType>
      </Control>
  ...
    </L_Controls>
  </window>
  <navigation>
    <Nav1>
      <source>Notepad</source>
      <destination>Find</destination>
      <steps>
        <Nsteps>6</Nsteps>
        <0>setText 15document = "a"</0>
        <1>FocusChange Edit</1>
        <2>Mouse Click</2>
        <3>FocusChange Find</3>
        <4>Mouse Click</4>
        <5>FocusChange FindWhat</5>
      </steps>
    </Nav1>
    ...
  </navigation>
  <dependency>
    <dep1>
      <window>Find What</window>
      <source>textbox Find what:</source>
      <destination>button Find Next</destination>
      <action>setText("A", Find What:)</action>
```

```
      <property>Find Next.IsEnabled = true</property>
    </dep1>
    ...
  </dependency>
  ...
  </application>
```

After extracting all the structural information of the windows of Notepad and recorded the necessary steps to open new windows, the second phase of the algorithm tries to find different behaviors in the several windows.

One of the behaviors that the algorithm was capable to identify in the Find dialog was the enable/disable between the text box (Find what) and the button Find next, as it is possible to identify in Fig.3.

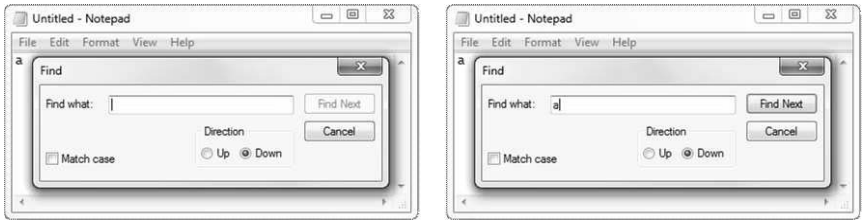


Fig.3. Enable/Disable Behavior in Find Dialog

The specification, in Spec#, generated to represent this behavior in shown below.

```
 [Action] void setTextFindWhat(string str)
modifies ButtonFindNext.IsEnabled;
{...}
```

In order to make the specification generated more readable, the name of the method follows the rule:

```
<Action><caption>
```

and the name of the GUI controls follows the rule:

```
<classeType><caption>+ '.' + property
```

In the method above, the name of the method is concatenation of setText (the action) and FindWhat (caption of the textbox control); the name of the GUI control is the concatenation of Button (the classType), FindNext (the caption) and IsEnabled (the property).

7 CONCLUSIONS AND FUTURE WORK

Automated GUI testing has become tremendously important as GUIs become progressively more complex and popular. One way to automate and systematize more the GUI testing process is to generate automatically test cases from GUI models. Our knowledge with GUI testing shows us that such models are very costly to be manually created and the specifications of software applications are rarely available in a way that models used by testing approaches can be automatically created from them.

We have presented a new technique that, through a reverse engineering process, allows obtaining a model of the GUI's structure and some of its behavior. This model is kept in a XML file from which a Spec# specification is generated for testing purposes.

The reverse engineering process proposed combines automatic with manual exploration which solves some of the "blocking problems" found in the approaches described in the state of the art section.

The algorithm used to infer the GUI's behavior follows a specific order of exploration that does not guarantee finding all possible dependencies among GUI controls. Dependencies may exist that show up only after a specific sequence of user actions different from the sequence used by the exploration process.

In the future, it is our intention to implement new algorithms to extend the set of dependencies among GUI objects found automatically by the tool (such as, dependencies that involve creating GUI objects); and improve the Spec# produced making it more complete.

REFERENCES

1. Chikofsky, E.J. and J.H. Cross II, *Reverse Engineering and Design Recovery: A Taxonomy*, in *Software, IEEE*. 1990. p. 13-17.
2. Systa, T., *On the relationships between static and dynamic models in reverse engineering Java software.* Reverse Engineering - Working Conference Proceedings, 1999: p. 304-313.
3. Moore, M.M. *Rule-based detection for reverse engineering user interfaces*. in *Reverse Engineering, 1996., Proceedings of the Third Working Conference on*. 1996.
4. Mori, G., F. Paterno, and C. Santoro, *CTTE: support for developing and analyzing task models for interactive system design.* Software Engineering, IEEE Transactions on, 2002. **28**(8): p. 797-813.
5. Csaba, L., *Experience with User Interface Reengineering Transferring DOS Panels to Windows*, in *Proceedings of the 1st Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '97)*. 1997, IEEE Computer Society.
6. Stroulia, E., et al., *Reverse Engineering Legacy Interfaces: An Interaction-Driven Approach*, in *Proceedings of the Sixth Working Conference on Reverse Engineering*. 1999, IEEE Computer Society.

7. Ricca, F., P. Tonella, and I.D. Baxter. *Restructuring Web applications via transformation rules*. in *Source Code Analysis and Manipulation, 2001. Proceedings. First IEEE International Workshop on*. 2001.
8. Barnett, M., Rustan, and W. Schulte, *The Spec# Programming System: An Overview*.
9. Veanes, M., et al., *Model-based testing of object-oriented reactive systems with spec explorer*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2008. **4949 LNCS**: p. 39-76, ISBN 978-3-540-78916-1.
10. Paiva, A.C.R., *Automated Specification-Based Testing of Graphical User Interfaces.* Department of Electrical and Computer Engineering, 2007. **Ph.D**.
11. Microsoft. *UI Automation*. msdn 2009 [cited 2009]; Available from: http://msdn.microsoft.com/en-us/accessibility/bb892133.aspx

Processamento de Dados, Informação e Linguagem

Information Extraction tasks: a survey

Gonçalo Simões, Helena Galhardas, Luísa Coheur

Instituto Superior Técnico, INESC-ID, DMIR, L2F

Abstract. An Information Extraction activity is a complex process that can be decomposed into several tasks. This decomposition brings the following advantages: *(i)* for each task it becomes possible to choose the best technique independently from the other tasks; *(ii)* an Information Extraction program can be developed as a set of independent modules (one for each task), making it easy to perform local debugging; *(iii)* it becomes easy to customize the Information Extraction activity through reordering, selection or even composition the tasks.
This paper presents a commonly used decomposition of the Information Extraction activities and gives detail about the most used machine learning and rule-based techniques for each task.

Keywords: Information Extraction, Natural Language Processing, Machine Learning

1 Introduction

With the increasing volume of publicly available information, companies need to develop processes for mining information that may be vital for their business. Unfortunately, much of this information is presented in the form of unstructured or semi-structured texts. Software tools are not able to analyze such texts and humans would take so much time to perform this task that the information would become obsolete by the time it was available.

Information Extraction emerged as a solution to deal with this problem. According to (Cowie & Lehnert, 1996), "*Information Extraction* starts with a collection of texts, then transforms them into information that is more readily digested and analyzed. It isolates relevant text fragments, extracts relevant information from the fragments, and then pieces together the targeted information in a coherent framework".

An *Information Extraction* activity can be very complex. Thus, it is common to decompose it into several tasks. This decomposition offers some advantages. First, it is possible to choose, for each task, the techniques and algorithms that better fit the objective of a particular application. Second, it is easy to locally debug an *Information Extraction* program since the module responsible for each task is completely independent from the others. Finally, an *Information Extraction* can be customized activity according to an application's needs, by reordering, selecting and composing some of the tasks.

This paper presents a possible decomposition of the *Information Extraction* activity in tasks. This decomposition is based on the work of (McCallum, 2005).

The considered tasks are: Segmentation, Classification, Association, Normalization and Correference Resolution. For each of these tasks we explain what is its purpose and which techniques can be used. An extended version of this paper can be found in (Simões, Galhardas, & Coheur, 2009).

2 Segmentation

The *Segmentation* task divides the text into atomic elements, called segments or tokens. Even though this task is simplified for Western languages due to the existence of whitespaces separating words, there are some cases in which simple whitespace separation may not be enough (Santos, 2002). Usually, segmentation for these cases is performed using rules that show how to handle each case.

The major problems related to this task can be found in oriental languages. For example, the Chinese does not have whitespaces between words (Li Haizhou & Zhiwei, 1998). For this reason, solving the problems described above is not enough in this language. In these cases, it is typically necessary to use external resources. Lexicons and grammars can also be used in order to accomplish the task of segmentation using syntactic or lexical analysis. Another approach for segmentation in Chinese uses techniques based on statistics. An example is the system described in (Li Haizhou & Zhiwei, 1998), which uses *N-grams* and the *Viterbi algorithm* (Forney, 1973) applied to segmentation.

3 Classification

The *Classification* task determines the type of each segment obtained in the segmentation task. In other words, it determines the field of the output data structure where the input segment fits. The result of this task is the classification of a set of segments as entities, which are elements of a given class potentially relevant for the extraction domain.

The rule-based techniques used in the classification task are usually based on linguistic resources, such as lexicons and grammars (Farmakiotou et al., 2000).

One of the most popular approaches to undertake classification is machine learning. Machine learning techniques used in this task are usually supervised, which means that an annotated corpus is needed. Five of the most common supervised learning techniques are the Hidden Markov Models (HMM), Maximum Entropy Markov Models (MEMM) (McCallum, Freitag, & Pereira, 2000), Conditional Random Fields (CRF) (Lafferty, McCallum, & Pereira, 2001), Support Vector Machines (Isozaki & Kazawa, 2002) and Decision Trees (Sekine, Grishman, & Shinnou, 1998).

4 Association

The *association* task seeks to find how the different entities found in the classification task are related. The systems that perform extraction of relationships are less common than the ones that perform the classification task (McCallum, 2005). This happens due to the difficulty in achieving good results in this task.

Many techniques in the association task are based on rules. The simplest approach uses patterns to extract a limited set of relationships. A more generic

rule-based approach for association is based on *syntactic analysis*. Often, the relationships that we want to extract are grammatical relationships (Grishman, 1997). For example, a verb may indicate a relationship between two entities.

The association task can also use machine learning techniques. One of the first machine learning approaches was based on probabilistic context-free grammars (Miller et al., 1998). These grammars differ from regular context-free grammars, because they have a probability value associated to each rule. When the syntactic analysis is undertaken, it is possible to find many syntactic trees. By using probabilistic rules, the probability of each tree is computed and the most probable tree is chosen.

5 Normalization and Correference Resolution

Normalization and Correference resolution are the less generic tasks of the *Information Extraction* process (Applet & Israel, 1999), since they use heuristics and rules that are specific to the data domain.

The *normalization* task is required because some information types do not conform to a standard format. This task is typically achieved through the use of conversion rules that produce a standard format previously chosen.

Correference arises whenever the same real world entity is referred in different ways in a text fragment. This problem may arise due to the use of: *(i)* different names describing the same entity (e.g., the entity "Bill Gates" can be found in the text as "William Gates"), *(ii)* classification expressions (e.g., a few years ago, "Bill Gates" was referred as "the world's richest man"), *(iii)* pronouns (e.g., in the sequence of sentences "Bill Gates is the world's richest man. He was a founder of Microsoft", the pronoun "He" refers to "Bill Gates").

Rule-based approaches for correference usually take into account semantic information about entities. A machine leaning approach for correference resolution is described in (Cardie & Wagstaff, 1999). This approach is based on clustering algorithms for grouping similar entities.

6 Conclusions

In this paper, we presented the decomposition of an *Information Extraction* activity into tasks and referred the techniques that are commonly used for each task. The goal of this paper is to supply material that can be used for the conceptualization of an *Information Extraction* program.

References

Applet, D., & Israel, D. (1999). Introduction to information extraction technology. In *Procedings of the* 16^{th} *International Joint Conference on Artificial Intelligence.* Stockholm, Sweden.

Cardie, C., & Wagstaff, K. (1999). Noun phrase coreference as clustering. In *Proceedings of the Joint Sigdat Conference on empirical methods in natural language processing and very large corpora* (pp. 82–89). New Brunswick, NJ, USA.

Cowie, J., & Lehnert, W. (1996). Information extraction. In *Special natural language processing issue of the communications of the ACM* (Vol. 39, pp. 80–91). New York, NY, USA.

Farmakiotou, D., Karkaletsis, V., Koutsias, J., Sigletos, G., Spyropoulos, C. D., & Stamatopoulos, P. (2000). Rule-based named entity recognition for greek financial texts. In *Proceedings of the Workshop on Computational Lexicography and Multimedia Dictionaries (COMLEX 2000)* (pp. 75–78). Pyrgos, Greece.

Forney, G. D. (1973). The Viterbi algorithm. In *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)* (Vol. 61, pp. 268–278).

Grishman, R. (1997). Information extraction: techniques and challenges. In *Information Extraction International Summer School SCIE-97* (pp. 10–27). Frascati, Italy.

Isozaki, H., & Kazawa, H. (2002). Efficient support vector classifiers for named entity recognition. In *Proceedings of the* 19^{th} *International Conference on Computational Linguistics (COLING02)* (pp. 390–396). Taipei, Taiwan.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the* 18^{th} *International Conference on Machine Learning (ICML 2001)* (pp. 282–289). Williamstown, MA, USA.

Li Haizhou, B. S., & Zhiwei, L. (1998). Chinese sentence tokenization using viterbi decoder. In *Proceedings of the International Symposium on Chinese Spoken Language Processing (ISCSLP 1998).* Singapore.

McCallum, A. (2005). Information extraction: Distilling structured data from unstructured text. In *ACM Queue* (Vol. 3, pp. 48–57). New York, NY, USA.

McCallum, A., Freitag, D., & Pereira, F. (2000). Maximum entropy markov models for information extraction and segmentation. In *Procedings of the* 17^{th} *International Conference on Machine Learning (ICML 2000)* (pp. 591–598). Stanford, CA, USA.

Miller, S., Crystal, M., Fox, H., Ramshaw, L., Schwartz, R., Stone, R., et al. (1998). Algorithms that learn to extract information–BBN: Description of the SIFT system as used for MUC-7. In *Proceedings of the* 7^{th} *Message Understanding Conference (MUC-7)* (pp. 75–89). San Francisco, CA, USA.

Santos, M. (2002). *Extraindo regras de associação a partir de textos.* Mestrado em Informática Aplicada, Pontifícia Universidade Católica do Paraná, Curitiba, Brasil.

Sekine, S., Grishman, R., & Shinnou, H. (1998). A decision tree method for finding and classifying names in japanese texts. In *Proceedings of the* 6^{th} *Workshop on Very Large Corpora (WVLC-98)* (pp. 171–178). Montreal, Canada.

Simões, G., Galhardas, H., & Coheur, L. (2009). *Information extraction tasks: a survey* (INESC–ID technical report No. 37/2009). Lisbon, Portugal.

Segurança de Computadores e Comunicações

Virtual Health Card System

Tiago Pedrosa[1,2], Carlos Costa[2], Rui Pedro Lopes[1], and José Luís Oliveira[2]

[1] Polytechnic Institute of Bragança, Portugal
[2] University of Aveiro, IEETA, Portugal

Abstract Electronic Health Records are key components to an efficient exploitation of information technologies in health care institutions. Nevertheless, several barriers that hinder its wide adoption subsists. The co-existence of dissimilar and incompatible health information systems and the absence of a unique central repository for personal medical data are some examples. This paper, we propose a web-based health records repository that allow citizens to have a unique virtual card that integrates all their personal clinical data. Privacy policies and the access control mechanisms are also discussed.

Resumo Os registos médicos electrónicos são fundamentais para uma utilização eficiente das tecnologias de informação em instituições de saúde. No entanto, existem várias barreiras que atrasam a sua implementação em larga escala. A co-existência de sistemas de informação de saúde díspares e incompatíveis, bem como a falta de um repositório central de informação médica pessoal, são alguns dos exemplos. Propomos um repositório médico, baseado em tecnologia web, disponibiliza aos cidadãos um cartão virtual único que agrega toda a sua informação médica. Questões relacionadas com a privacidade e mecanismos de controlo de acesso são também abordados.

1 Introduction

The use of Information Technologies (IT) to support health care services is a reality commonly spread in the society. Nevertheless, one of the biggest challenges in the health informatics is still the creation of an Integrated Electronic Health Record (IEHR) - a patient longitudinal record that aggregates all generated information in clinical consultations. The balance between privacy and accessibility issues is one of the reasons that makes health care a rich scenario for security technologies [8]. On one hand, we must enforce the privacy of sensible information and, on the other, the quality of healthcare services demands sharing and remote access to patient information [9,7].

The IEHR is an important tool that clinicians can use to be better informed about patients' medical history. Due to changes in citizens way of living, it's

normal one person to have several clinical appointments in different cities, regions and even countries. Citizens move their residence during lifetime, travel more regularly, for working, for leisure or even for medical care [12]. Hence, the information generated will be disperse along several institutional information systems. A unique view of the disperse EHR, would improve the quality of healthcare services. To create an integrated access to the information that is disperse amount several systems, a single patient identifier should be necessary to simplify the aggregation of medical data. However, this isn't a straightforward task since each patient may have different identifiers in various systems.

As the information is spread in several organizations, its sharing has to respect rigid laws and regulations that makes difficult the IEHR implementation. Other difficulty is based on the lack of well established communication standards between different EHRs systems, despite some efforts [11].

2 Materials and methods

In most scenarios, the regulatory and law framework for sharing health records can be satisfied imposing the patient informed consent (several models use this approach [5,1,3]). As the patient is entitled to request a copy of its records and share with anyone that he decides, the sharing is make using his consent instead of the organization that stores the records.

In those heterogeneous environment, involving different organizations, public or private, a secure authentication mechanism is mandatory. In the last decade, the use of smart-cards in healthcare information systems has been consensual, as they provide a secure way for storing information and authentication credentials for remote authentication [6]. The Electronic Health Card (EHC) is basically a smart-card that is used for saving useful information for administrative tasks, emergency medical information, security certificates and, in some cases, e-prescriptions. This type of tokens are used in some countries like, for instance, Germany and Austria to achieve a national IEHR solution.

As discussed, the IEHR implementations needs to provide an integrated access mechanism to disperse information. So, the integrator system must know the data location, more precisely the query engine service to extract information of a specific patient. This linkage information can be stored in the integrator database, however some projects decided to extend electronic health card to support that service. Hence, the Virtual Unique Electronic Patient Card (VU-EPR) appears as a possible solution [4].

The VU-EPR is based on a token containing card-owner resident clinic-admin information, as well as structured references to its electronic records. The smart card securely contains this reference structured data set. The implementation of Public Keys Cryptography and Crypto Smart Cards, unequivocally provides a way to securely store, transport and access the card-owner information. Moreover, it also grants the owner full control over the access to its data, through a Pin and/or biometric registration. Finally, it also allows the card-owner to entitle information access levels to other users such as the clinical professionals. The

main benefits associated to this solution can be characterized by highly scattered geographical storage requirements. This model empowers patient enabling the discretionary access to remote data, when crossed VU-EPR card with health professional card, and allows an open access to the medical emergency data stored in the card. Upon this model, we developed a VU-EPR solution named Multi-Service Patient Data Card (MS-PDC). The MS-PDC was modeled to provide five complementary services and results from an extension of a first developed (VU-EPR) model exclusively oriented to Electronic Patient Records [5,4] information: i) Administrative data support; ii) Emergency Clinical data support; iii) Hyperlink base, build upon the URL schema and that allow to link the patient clinical and genetic distributed information; iv) Patient digital credentials support and management; v) PDC owner verification capability.

The MS-PDC uses URLs to fetch the information on the disperse systems and present them to the user as a unique view of all distributed data. This model copes well with mobility issues, such as the gathering of disperse data and controlling the access to it. Nevertheless, in a wider concept of mobility it's not feasible that all patients will hold the same type of card world-widely. Other discussable aspect is the need of the presence of the physical card in the system whenever exists the need to access the patient EHR. Thus, we propose a model where a system will hold the card in behalf of the user.

3 Results

The Virtual Health Card System (VHCS) proposed, appears as a solution to overcome the drawbacks associated with physical token dependency. Instead of the EHC being hold by the patient (Figure 1), it will be hold by a service (Figure 2). The service will store and provide access to the EHC when requested and only if the authorization is granted by the predefined policies. Therefore the information can be used after patient informed consent, since the links to the information will be on the system. The informed consent will be also stored in the card, visible to other system components that can read it and apply that policies controlling the access to the patient information. Moreover, other important advantages could be identified in this proposal. First, it expedite the processes of backup and revalidation of credentials. Secondly, an important issue, it allows the dynamic update of links on the patient card whenever new information is created. Most of this features became available due to the continuos presence of the card in the system(Figure 2).

This approach will permit the disassociation between the credentials used by users in system authentication and the credentials used inside the system. For accessing to his EHC, the user will authenticate using a token. The system is sufficiently flexible to support different tokens including the new Portuguese Citizen Card, an electronic identification card (eID card) that contains a certificate for authentication. Moreover, if the user token or eID is lost or stolen, the system can temporary block the access to the Virtual Health Card until the new token be available and associated to a patient Virtual Health Card.

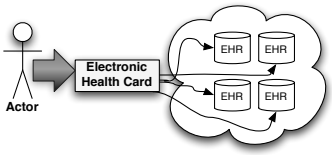


Figure 1. Patient hold his electronic health card

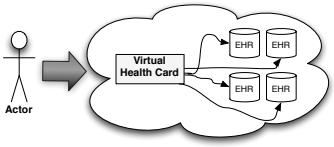


Figure 2. System holds patient electronic health card

Our proposed model (Figure 3) is composed by 4 main components: the credentials, the access policy and two types of Universal Unique Identifiers (UUIDs). The credential component is responsible for securely storing the private and pub-

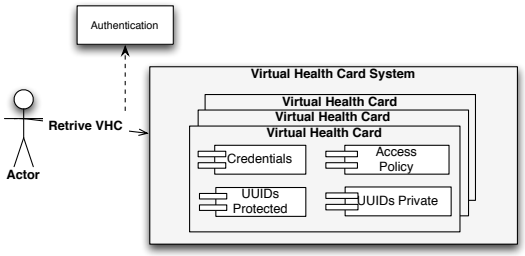


Figure 3. Virtual Health Card System

lic key of the user. The access to the private-key container is only available to the authenticated user (the actor), by the way of a secret (could be a password or other method [2]). The private-key inside the container is the credential that will be used internally for authentication, signing, cypher and de-cypher the information. This modus operandi separates the credentials for authentication in the system from the credentials that the user will use to logon.

In access policy component, the patient defines the informed consent to his information, identifying healthcare professional and granting specific permissions. Each new entry is signed with the patient's private-key. Hence, each time a system component makes a request to access the patient information, the system checks if the requesting user has the necessary privileges (through the access policy), verifying always if the policy was really signed by the patient.

The UUIDs represent indexes to the disperse patient information. The VHCS use this information to create an unique view of it IEHR. These universal unique identifiers will work as links to the remote information. Moreover, each link has also complementary information about access mechanisms (or services). The system provides two types of UUIDs spaces, enabling a patient private links area (i.e. UUID Private) to have information that only him can reference and other

area (UUID Protected) where are placed the references accessible to specific healthcare professionals.

The Private UUID may be used to handle references of very sensible and discriminatory information. On this component, the patient can manage the information that he does not want accessible to any health professionals, in any occasion. To enforce this behavior, the system will cypher the references with the users correspondent public-key forcing that only with the private-key of the user this information could be read. The access to this private information demands always the explicit patient consent.

The Protected UUID is the place where other system components (or external services) can update the UUIDs, as new information is being produced in several health systems. Components that in the behalf of an authenticated and authorized user want to access the patient's information, query this component to get information about remote patient data location and how to access it.

The credential component will ensure that only an authorized user can access his private-key and that Private UUIDs will only be accessible using the correct private-key that is stored inside the credential component. It's also propose that the Protected UUID and the Access Policy be cyphered using a system key, that must be shared between the components that need to communicate with the VHCS. Therefore a component is obligated to register in the system to obtain the access key.

In a emergency scenario, the patient could not be able to provide the intend consent. The VHCS is prepared to handle this situation granting the practitioner access to the patient EHR and bypassing the access policy for the UUID Protected. This "break-the-glass" mechanism will only give access to information that isn't protect in the private area (UUID Private component). This mechanism will generate auditing records for future analyze and detection of misconducted access.

The VHCS proposal provides an important indexing and retrieve service of disperse information and the access control mechanisms to ensure the patient data privacy and confidentiality. It will also enable users to have an unique authentication in the system, providing a single-sign-on behavior. This service will present the user credentials to other components as needed.

4 Conclusions

The proposed model copes well with requirements of mobile citizens EHRs, it separate the credentials used for authentication from the credentials used in the indexing system. It enables the creation of a dynamic mechanism to update references of remote patient information. It also copes with the existence of different identifiers for the same patient, along different healthcare systems. Moreover, it empowers patients with the capability to decide what information is absolutely private from all the information that exists in the disperse EHRs. Finally, it implements an informed consent mechanism that respects the regula-

tory framework for sharing of healthcare records between distinct professionals (or institutions) in different regions or countries.

Further work will be need namely to decide how to implement the access control policy. The idea is to have a central RBAC policy database [10] that defines the permissions for each role like, for example, the permissions to a practitioner or the permissions to a nurse. With the central RBAC policy database the patient only have to decide which role he grants to each professional profile in the access policy component.

References

1. Ankica Babic, Carlos Costa, José Luís Oliveira, Natalja Voznuka, Ilidio Oliveira, Markus Storm, Victor Maojo, Fernando Sanches, Miguel Santos, Antonio Sousa Pereira. *Confidentiality and security issues in web services managing patient clinical and genetic data.* Linkopings, Sweeden, 2004.
2. J. Basney, M. Humphrey, and V. Welch. The MyProxy online credential repository. *Software: Practice and Experience*, 35(9):801–816, 2005.
3. J. Bergmann, O. Bott, D. Pretschner, and R. Haux. An e-consent-based shared EHR system architecture for integrated healthcare networks. *International Journal of Medical Informatics*, 76(2-3):130–136, 2007.
4. A. S. Carlos Costa, José Luís Oliveira. Electronic patient record virtually unique based on a crypto smart card. *Lecture Notes in Computer Science*, Volume 2722/2003, 2003.
5. A. S. V. G. R. Carlos Costa, José Luís Oliveira. A new concept for an integrated Healthcare Access Model. *Studies in health technology and informatics*, 95:101, 2003.
6. H. Chien, J. Jan, and Y. Tseng. An Efficient and Practical Solution to Remote Authentication: Smart Card. *Computers & Security*, 21(4):372–375, 2002.
7. F. Colasanti. ICT for Health and i2010-Transforming the European healthcare landscape-Towards a strategy for ICT for Health [online]. June 2006. Luxembourg, 2006.
8. C. Dalton. The NHS as a proving ground for cryptosystems. *Information Security Technical Report*, 8(3):73–88, 2003.
9. K. D. Mandl, P. Szolovits, I. S. Kohane, D. Markwell, and R. MacDonald. Public standards and patients' control: how to keep electronic medical records accessible but private commentary: Open approaches to electronic patient records commentary: A patient's viewpoint. *BMJ*, 322(7281):283–287, 2001.
10. J. Reid, I. Cheong, M. Henricksen, and J. Smith. A Novel Use of RBAC to Protect Privacy in Distributed Health Care Information Systems. *Proceedings of the Eighth Australasian Conference on Information Security and Privacy (ACISP 2003), LNCS*, 2727:403–415, 2003.
11. Technical Committee ISO/TC 215. Health informatics — electronic health record — definition, scope, and context - iso/tr 20514:2005(e). Technical report, International Organization for Standardization, 2005.
12. The European Economic and Social Committee and the comittee of regions. Final report on the implementation of the commission's action plan for skills and mobility com(2002) 72 final. Technical report, Commission of the European Communities, 2007.

Índice de Autores