UbiLang: Towards a Domain Specific Modeling Language for Specification of Ubiquitous Games

Ricardo Guerreiro, André Rosa, Vasco Sousa, Vasco Amaral, Nuno Correia

Computer Science Department, Faculdade de Ciências e Tecnologia – Universidade Nova de Lisboa, Quinta da Torre, 2829-516 Caparica, Portugal
{rmg15404, adr13041}@fct.unl.pt, vasco.sousa@gmail.com {vasco.amaral, nmc}@di.fct.unl.pt

Abstract. As new ubiquitous projects emerge, it is often required the integration of ubiquitous devices in development frameworks. This commonly leads to developing new frameworks, usually created in General Purpose Languages (GPL). Although this solves immediate problems, it also leads to a decrease of productivity and efficiency, due time spent while adapting code. This results, in most cases, on a development process starting from scratch when most of the times the concepts were already used in previous projects. This project proposes tackling this problem by implementing a Domain-Specific Modeling Language called UbiLang. By carefully taking into account concepts of the domain problem, UbiLang has the main goal of enabling ubiquitous games developers to speed-up their problem specification during the design phase. Allowing then early error detection by validating the system model on a higher abstraction level than code and by improving application development time contribute to faster application prototyping.

Keywords: Domain Specific Modeling Language, Ubiquitous Game Devices, Ubiquitous Gaming, Meta-Modeling, Language Engineering.

1 Introduction

The culture of gaming has a long tradition since ancient board games, such as the Mesopotamian's "Royal Game of Ur"[1]. Currently, games are a real worldwide phenomenon recognized as an important research topics[2] and almost every day new devices are released into market with the purpose of improving gaming experience. Permitting the five human senses to participate in the experience, these improvements have the goal of providing new perspectives of time, space and interaction within the game itself. This technological development can be observed in the evolution of ubiquitous hardware like gloves, head-mounted displays and others, which are often tested in games[2].

As defined in [3], a Ubiquitous Device is – "*an electronic device capable of using its internet, wireless and other networking capabilities that are so embedded in the environment that the devices can be used virtually used anywhere and anytime. This concept embraces a broad range of possibilities, which include communications (cell phones),ubiquitous computing (notebook computers), delivery of images (displays) and products for identifying or managing people and things (objects using wireless IC tags, like RFID tags)*".

Another fact its that the time to market is of a crucial essence in the Ubicomp domain area, due its quick evolution pace. As new Ubiquitous Projects emerge, it is often required the creation of a new types of components (forming new Ubiquitous Devices).

This leads to the necessity of development of new suitable frameworks (that support the project's development) normally programmed in a General Purpose Languages (GPL), such as C++, C# or Java. Although these approaches tend to solve immediate problems, they also lead to a decrease of productivity, due to the low-level code re-use of solutions that are very similar. This compels, in most cases, to development from scratch, as already analyzed in [4].

At the same time, the domain expert hardware/game developer would benefit from a language with a higher level of abstraction that would allow the specification of models using domain terminology instead of a language from the solution domain (e.g., C++ or Java). The last situation of using code, potentially leads to semantic gap problems (i.e., the way we think about the problem in terms of games, has to be twisted, in an error prone fashion, to the way it is expressed in computational terms).

A Domain-Specific Modeling Language (DSML) is according to [5]: a language "*used to make specifications that manual programmers would treat as source code and if formed correctly it should apply terms and concepts of a particular problem domain*". We propose a DSML to solve the referred problems, by allowing the development of applications with Top-down (Designers to Programmers) or Bottom-up (Programmers to Designers) paradigm views.

To accomplish this we will, in section 2, engage in a domain analysis of the Ubicomp topic by exploring reusable aspects of the hardware components and correspondent behaviors. After that, a design of the DSML will be made, which is expected to provide the domain expert with faster means for lowering error prone model specification in section 2 and 3. This will be achieved by dividing the gained knowledge into two levels:

- **I.** **Structural Level**: where all the hardware components and the most used virtual objects properties/aspects are gathered;
- **II.** **Behavior Level**: where the most used application behavior transitions and states are gathered;

Finally, in section 4, a solution will be provided using a real life case-study of a wearable interface developed by the Multimedia Group at FCT-UNL dubbed as Gauntlet [6] and an application example for the Gauntlet, named Noon [6, 7]. We will then validate our statements of productivity increase and usability through an empirical study, using domain experts as subjects. In section 5 we conclude our work.

2 Domain Background

Ubiquitous Projects belong to the Ubiquitous Computing (Ubicomp, for short) domain area and, depending on the context, to the larger Multimedia base domain area. This allows in addition the Ubicomp are to be fused with another domain area, the Augmented Reality (AR), due to the latter using the same technology developed in Ubicomp to expand reality with computer interaction. These domain areas, similar as they are, can have the same development background. So the next sub-sections will introduce the most relevant development methods for these domain areas and conclude by presenting the domain analysis made with the Ubiquitous Projects.

2.1 Augmented Reality

As stated in [8]: "*AR is a variation of Virtual Environment (VE), or more commonly known as Virtual Reality (VR)*". A VR differs from an AR, as the first merges the user perspective with the artificial world, therefore hiding the real world. The AR approach uses computer generated elements to complement reality in real-time. To achieve results like this, an AR project can use Ubicomp technology (such as a PC with a camera and a specific pattern, for example) and build applications able to generate these VEs. These applications are then called AR applications and are commonly developed using AR frameworks, which have the ability to manage the particular Ubiquitous Devices used in the application. Other developments methods can consist on Visual Programming Languages (VPL) and GPLs. These three concepts are going to be further discussed.

Augmented Reality Frameworks, can go from software libraries like ARToolkit [9] to concepts of collaborating distributed services like the Distributed Wearable Augmented Reality Framework (Dwarf for short) case [10, 11]. Still, frameworks like ARToolkit, due to the fact of being “software libraries”[12], when compared with Domain-Specific Modeling (DSM) or VPLs, can miss the major advantages of the fast application prototyping inherit in these latter approaches. It is, for example, also surpassed by the Dwarf framework *services*, however as a gain, its modules have already been validated and tested and therefore can be reused/applied in a possible future Domain Specific Modeling (DSM) Generator approach. In Dwarf’s architecture, because of its concepts of collaborating distributed “*services*” that can be developed in a wide range of programming languages (e.g. Java, C++, Python), give the framework the power of platform/programming language independency. These services consist in collections of interdependent modules of code that have a set of requirements called “*Needs*” and capacities called “*Abilities*”. Each of the modules is then connected with other modules within a network, forming groups that are then controlled by a special “*service*” named Service Manager. This “cluster” concept is similar to an element abstraction in a DSML, yet it lacks the domain specification possible by such elements as it does not reuse the code as a DSM Generator would do, thus keeping the same issues when compared with GPL approaches.

Visual Programming Languages (VPLs), are programming languages that let programmers create new applications by graphically manipulating program modules. A VPL, to achieve that purpose, uses visual expressions, spatial arrangements of text and/or graphic symbols, rather than specifying them in a textual manner. So a VPL oriented environment results in a language with an inherent visual expression for which there is no obvious textual equivalent [13]. They are also associated with specific applications or frameworks, with some examples like: Max/MSP/Jitter [14], Pure Data [15], and others [4].

Media Processing Frameworks is a type of development where the programming is still being made in a textual manner using GPLs for that purpose. ARToolkit, for example, can also be classified as a Media Processing framework or software library that uses GPL. Some widely used examples are Processing [16] and openFrameworks [17]. Both present a simplified interface to powerful libraries for media, hardware and communication interaction.

These solutions, despite solving the prompted issues (multimedia object manipulation, for example), do not address all Ubicomp problems (as the majority of the Ubiquitous Projects do not take a framework approach to solve their problem, but instead use GPL programming). This divergence comes as the Ubicomp domain area and the AR domain area can be fused, there are some components/requirements or actions in Ubiquitous Projects that are not commonly taken into the AR domain scope. Also, in Ubiquitous Projects, there exists a tendency to develop new types of devices, from different off-the-shelf components with the main objective of innovating how players interact within a particular game. AR Projects instead, normally take already developed devices and just enhance the user's perception of reality. In the VPLs case, being similar to a Graphic DSMLs as they are, the biggest discrepancy comes from the capability of a DSML to be built specifically to a particular domain. Finally, GPLs and Media Processing Frameworks case still have reusability issues from one application to another together with possible code errors or/and inconsistencies between both programmers and application designers visions for the final application. Still this analysis demonstrated the most common components and functionalities currently available, providing an idea of the ones that were most used or required by the AR community in general.

2.2 Survey of Projects

Presently there is a wide plethora of Ubiquitous Projects and some of them were analyzed in surveys like [2, 6]. With the objective of widening the domain scope to demonstrate the usability of the proposed DSML in a wide range of Ubiquitous Projects, a selection parameter was required to analyze these projects. The analysis began by searching for similarities between projects. This lead to two major comparison parameters: (1) the hardware components used to build the particular Ubiquitous Project and (2) the way these components were used or the behavior they add in the program workflow. With the first comparison parameter it was possible to restrict the analyzed projects to a much smaller number. The second parameter came when even, repeating some hardware components, the behaviors each of them had (in their respective projects) was different from one application to another. This gave not only different and new behavioral analysis (which was useful to be generalized) but decreased even further the number of projects to be analyzed. With these criteria in mind, we have proceeded to a discussion and analysis found in [18] of the various selected projects that were found to incorporate the widest array of components and behaviours: 6th Sense [19], Blinkenlights [20], Brainball [21], InStory [22], Pirates! [23], Uncle Roy All Around You [24], Epidemic Menace [25], Noon [6] and Headbanger Hero[26] .

The analysis then gave an wide range of the commonly used components in the domain area. They were then added to the lowest levels of the Structural Level and soon it was clear that, in order to offer some type of organization at modeling time, the components were needed to be divided in several categories, as seen in Fig. 1. This allowed programmers/hardware developers to rapidly and easily construct devices and therefore a concept of a "Virtual Device" was added to the language.

Within each of the categories, shown in Fig. 1, it was put the identified elements of the analysis. For example, in the "Visual" category it was needed a "Monitor", "Camera" and "Visual Effects" elements, which enabled the definition of a generalized input, output hardware component and also a property that gave effects to all visual components, (being also possible to define individually "Visual Effects" element within each defined "Monitor" and "Camera" element), the other categories were also filled with identified domain elements abstractions and more details can be found in [18].

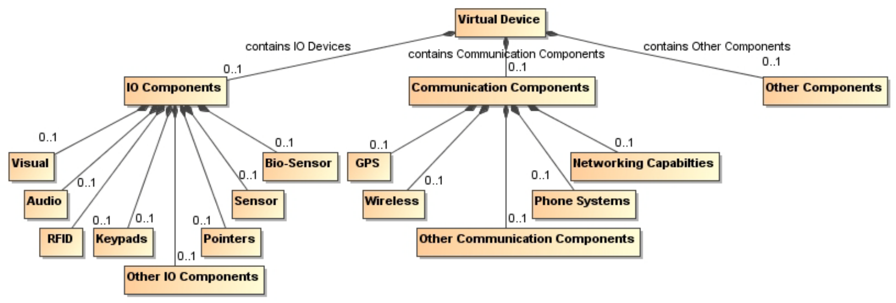

Figure 1. Virtual Device Compartments

Additionally to the components used in the projects, their contexts were also analyzed, resulting in some relevant concepts. One of these, was that the need (in the majority of the cases) to interact with virtual objects which permitted, for example, interaction with physical hardware components/objects external to the application itself (audio, video, textual, video file, or even a model generated by an outside application). With this notion, the Structural Level was completed by creating a new element called "Object" which stood at the same hierarchical level of the Virtual Device. To this "Object" element was given the possibility to define virtual representations, which could be referenced to a specific type of external multimedia or model representation files. Furthermore, it could have its own physical properties, in case of a NPC or human avatar, or even virtual values attributes like health, for example. The relation with the Virtual Device would then be made via its physical representations properties, as for example RFID Tags, real world coordinates or association with specific Virtual Devices.

From the projects, another identified requirement/concept was the need for (at modeling/application designing time), provide users with the opportunity to instantly work on the application's workflow without concerns and therefore rapidly express (in a language much closer to their own) a prototype of the application's behavior/context. So with this concept and the analysis already done, the Behavior Level began to be developed following a template of a finite state machine model, for all the identified actions [18].

2.3 Model Driven Development

In a Model Driven Development (MDD) approach, specifically in DSM, a model, for short, serves as a mean of visually representing abstractions of system concepts and features [5]. This model can be used for designing purposes (when the problem domain requirements are too complex to be expressed with code), to reverse engineer systems (by creating model-based visual documentation, which helps understand the capabilities of a system after it was already designed and built) and other functionalities. A metamodel is a DSML specification, which describes an abstraction level higher than the model, at the point that it abstracts the concepts used in the model or abstract syntax below. The metamodel has the purpose of describing and expressing concepts of a language, their properties, constructions and rules (like relationship, correctness or hierarchy rules between concepts) [4, 5, 27].

Domain Specific Modeling, has two aims: (1) raise the level of abstraction further than the current programming languages by specifying the solution using problem domain concepts; (2) generate final applications in a chosen programming language or other form of high-level specification [5].

The first objective specifies a Domain Specific Modeling Language, or DSML, which is a kind of typically declarative language that gives an expressive power to a particular problem domain, simplifying the development of generalized applications in these specialized domains using high-level abstractions of the domain concepts for that purpose. The definition process of a DSML, as stated in [5], consists of five main phases: (1) Domain Analysis where the problem domain necessities and/or requirements are identified and analyzed based on a number of different applications/systems that belong to the same domain for similar features or concepts; (2) Language Design where the domain analysis features gathered from the previous phase are formalized in the design of the DSML metamodel, (3) Language Implementation represents the definition of the visual representations of the DSML features; (4) Language Testing validates a DSML to multiple example cases and (5) Language Maintenance when the domain area evolves, leading to new requirements that will be used to update the language.

In the second objective the generation process can, normally, be supported by an application framework or API using a domain specific generator or a high level of abstraction. The idea behind a DSM generator comes from the notion that in DSM, application modelers do not expect the full code of the framework to be implemented but only the "code" they modeled in the DSM Editor instead [5, 28, 29].

DSM solutions are meant to be used when applications features or domain requirements have similarities. In these situations, application programmers tend to focus in their applications unique features development rather than reimplementing similar functionalities [5]. In addition, when comparing with other general purpose modeling languages, like UML [30] (Unified Modeling Language), the DSM process takes a large advantage as it can join all the diagrams needed in the UML development process, for instance, into a single metamodel. By the time the models are being implemented, in the UML process, they are made in a independent way from the designed models themselves. This can lead developers astray with questions of not properly specified aspects that are (possibly) not true in the application/system domain or were just simply ignored. Also it is virtually impossible to generate full application's code as the generality of the modeling language does not know anything of the application domain origins or its problems. In this way, in the DSM process, the metamodel, prevents semantic errors in illegal designs that do not follow the model architectural rules defined previously. Subsequently, the code generated from it does not contain logic errors, syntax or careless mistakes as it was specified by a DSM Generator (that was developed by an experience domain expert developer) [4, 5, 29, 31].

3 UbiLang

Taking the five domain phases described in the previous section, this chapter presents the work done in the development of UbiLang.

Domain Analysis, where the elements, classes and concepts were identified as requirements to be available in the DSML, this was done in Chapter 2 with the analysis in the various domain areas and projects. One these concepts, was the need of providing at modeling time, the two programming paradigm views (Bottom-Up (programmers to application designers) and Top-Bottom (vice-versa)). This came of the necessity to not constrain the creativity in the development process and allowing designers to work on the application behavior first and component configuration second (Top-Bottom view). Also the programmers could configure (if desired) each of their application components (hardware or virtual) and leave the application behavior formalization for afterwards (Bottom-Up). With this notion in mind, it was defined the two levels already explained, within the DSML: Structural Level and Behavior Level.

Design, the domain experts, independently from the development approach, categorized their applications between several categories (visual, audio, keypad, etc.) and behavior flows. With analysis done in Chapter 2 (previous phase) came too many elements/concepts, so a filter was added that abstracted several elements/actions into one. An example, already described, was the elements in the visual category, the other categories went through the same process and more details can be found in [18]. This type of definition was adopted in UbiLang, because it allowed an easier device's expression and greater creativity for programmers and designers alike, taking it to full extent levels without concerns about programming new hardware components/behavior flow with error-prone development.

Also the analysis acknowledged the most relevant properties required for each of the identified elements/connections (for configuration purposes). For example, in the specific comparison connections cases, it was identified the most common used comparison parameters. These parameters went from different source-target types to comparison conditions, as for example positional parameters between components and objects GPS position. Furthermore it was added the possibility, (for all connections types), to be delayed a certain amount of time desired by the modeler and displaying this information in each of the textual label's connections alongside its name[18].

The next Fig. 2 presents a general "metamodel" created from the previous explained phases, with the concerns for the modeler creativity, for each game it was added the possibility to organize it using several "Virtual Device Manager", "Object Manager" and "Behavior" elements. This gave the possibility to, for example, organize devices through teams (human and NPCs), virtual objects through specifications (avatars and virtual objects) and behaviors through specific actions (repeat actions and one time actions). The "LAN" element stood at the same level as the Virtual Device as it represented the network itself as well the Virtual Devices (they still required the "LAN Capability" element [18]) connected to it.

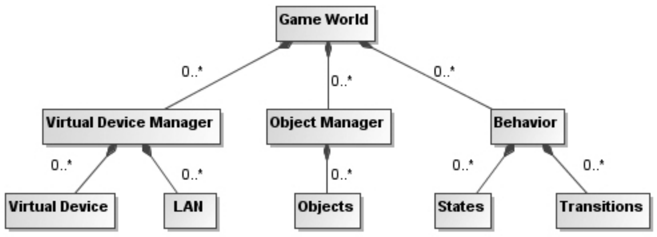

Figure 2. General UbiLang metamodel.

Implementation, the language was implemented using the workbench Eclipse with the aid of the Graphical Modeling Framework/Ecore Modeling Framework (GMF/EMF)[32] and its plug-ins. This workbench offered one of the best possibilities to customize elements interface layouts and other graphical interface improvements, when compared with another tools [29].

As seen in Fig. 4 to 5, the workbench offered the possibility of designing basic elements by displaying visual information in form of icons in addition to the textual labels. As well it offered a way to encapsulate information, in the form of compartments, this is helpful when the models tend to get bigger and more complex, as in the case of Fig.5. Also, for aiding the user, when building their model there was various assistants, like a customizable side palette toolbar that had all the elements, which the user could create its models. When hovering the canvas, or the inside of a compartment, a popup bar displayed the possible elements that could be inserted. In the Behavior model by clicking and hovering on the border of the icons/compartments, two arrows displayed (an arrow going out and another going in), and clicking in either one would displayed all the possible connections to/from the clicked element to an existent/new element. Also, in the Behavior model compartments, the state elements were grouped in 4 categories: General Actions (general application behavior as new threads, initialize a behavior or another specific behavior), General Object Actions (change configurations of an object, positions, etc), General LAN Actions (general networking actions as send files to all the groups in the network) and General Virtual Device Actions (change devices configurations). These compartments, for an easier use, when double-clicked on the border opened a new canvas that allowed an easier model design and when closed displayed the edited elements of the second canvas, in the inner compartment, with the opposite still possible. Finally, when right-clicking on each of the elements placed on the canvas (icons, connections or compartments) and displaying (if not already) the built-in Eclipse's "Properties View" plug-in, it was possible to change the properties identified of the domain analysis and associated with each of the elements[18].

This visual proposal [18] was found to be the most conformable to the objective of combining both programming view paradigms and also offer an easy and understandable way of designing ubiquitous game applications, thus allowing the creative process to not be hindered with programming issues.

4 Validation

Continuing the DSM process the next phase, validation, was done by modeling an application called Noon. Developed by Tiago Martins, the Noon's context introduced a long-story mystery were the player took the role of a detective [6, 7], using a device called Gauntlet, seen in Fig. 3, and another called Tome. This was one of the case-studies chosen to validate the language.

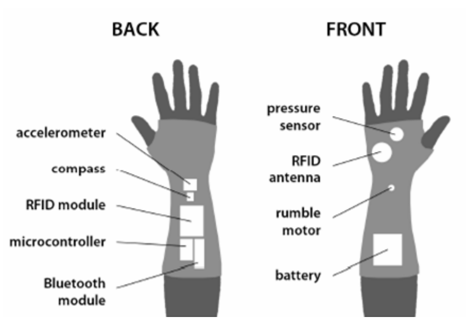

Figure 3. Gauntlet, extracted from [6].

Components, The choice of hardware technologies relied in a selection procedure based primarily on wearability factors (how the hardware influences motion responses, muscular shifting, the temperature it causes and the total weight it brings to the user) and secondly by how the sensors reacted to different types of environment. The final ubiquitous devices technology composition for the Noon framework was for the:

- Gauntlet - with a possible representation seen in UbiLang on Fig. 4 with an accelerometer, a digital compass (or magnetometer, which combined with the accelerometer can provide a more absolute measure of the user's tri-axial arm movement), a RFID Antenna and Module (combined can read and interpret RFID Tags), a Bluetooth module (used for communications with the Tome, a force resistance (or pressure) sensor, a rumble motor and a LED, for user feedback and a battery (for portability).

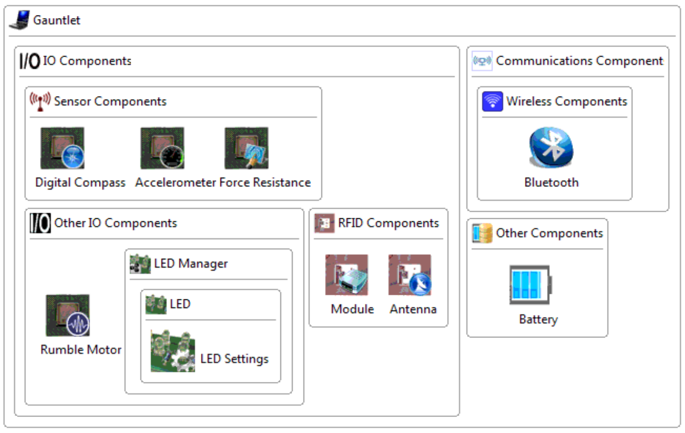

Figure 4 – A possible interpretation of the Gauntlet in UbiLang.

- Tome – this object can be any platform of Ubiquitous Computing technology (a notebook, a PDA, a smart-phone, etc.). The only mandatory requirement is a monitor, a set of speakers and a Bluetooth module for receiving Gauntlet's communications.
- Object Tags – Each of the in-game objects has a RFID Tag associated with it which the Gauntlet reacts upon.

Virtual Objects, within the game there were a total of six objects (associated with the physical tags mentioned above) that react with different time periods within the game depending on the direction where the Gauntlet is pointing: a snow globe, a cup, a picture, a schoolbook, a hammer and a table clock.

Behavior, the Noon application behavior starts for waiting any of the user input. If the user puts the Gauntlet in a vertical position, the accelerometer and digital compass detect this movement and display a visual feedback by turning on the Gauntlet's LED. The user can also make a horizontal movement, to which the application reacts by giving a clock ticking sound and permitting to change the "game time". The user, then uses the Gauntlet's ability of reading RFID tags (which are attached to physical objects), by means of the RFID Antenna. This allows "triggering memories" from a game character called Mrs. Novak, and display them on the Tome. In some of these objects, if it is detected a specific motion pattern, for example "shaking the snow globe" or "pouring the cup", the player is shown "deeper memories". In the process, a more "intense memory" triggers a Poltergeist, which the player must capture by listening to the sounds it makes, in order to continue the investigation and solve the mystery[7]. Noon is also an endless game so when all the memories have been read, the game restarts. Fig. 5 represents a proposal for a portion of the applications behavior (capture of the poltergeist) in UbiLang.

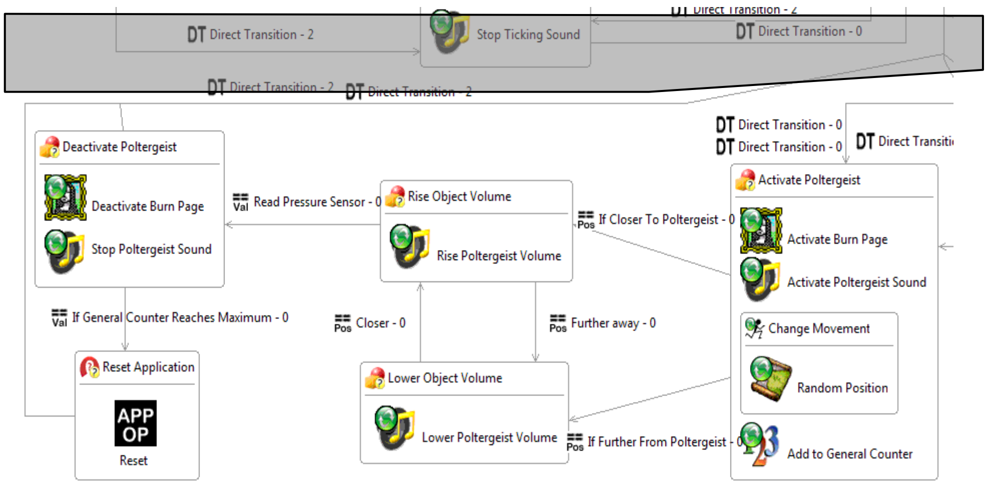

Figure 5. A partial view of an interpretation of the Noon Behavior

After sucessfully modeling a use-case the second part of the validation process was letting the domain experts test the language themselves. By collecting a group of 9 users, as advised in [33-35] and giving instructions to install the plugin, it was asked to them to complete a different exercise that had a different context from Noon, which they started in a provided tutorial and continued to answer a questionnaire. Some of the responses were:

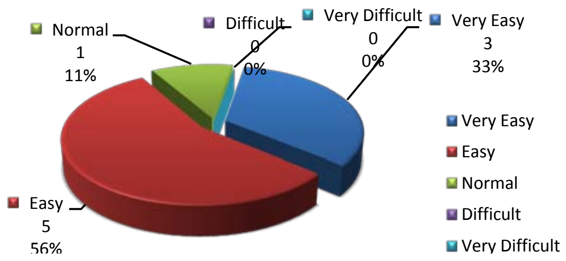

Figure 6 – "How easy it was to learn the UbiLang concepts?"

Other question "Do you feel UbiLang is a value-added compared to the previous application designing system?" the users fully agreed that it was a value-added to a coding development system alternative or adaptation of VPL's or Media Processing Frameworks. The follow-up question "Explain why?", provided such answers like: "*It might allow people with no imperative programming experience to experiment with behaviors and produce programs for Ubicomp interfaces*" and "*Using a visual representation allows a better understanding of the application and it is easier (less errors) and faster to build it. Furthermore by using models we gain various advantages, instant verification, possibility to simulate, automatically generate code and so on*".

5 Conclusions and Future Work

By means of case-studies and user assertions, we demonstrated that we reached our objectives and, in addition, the ubiquitous models maintenance time is also greatly shortened by the simple act of adding/removing a missing/existent component or action. The language is but a first step on the development as it is ready to be taken to a DSM Generator phase by assigning modules of validated code to the language elements and quickly pass from a designed model to executable code and therefore running application. Once we have a complete DSL, we can also explore model verification techniques to detect inconsistencies in the implementation already at design time. In what concerns the language editor, it would be interesting to enhance it so it would support new visualization modes and provide a better usability for the users. The main objective of developing this language was to provide an easier and faster way to increase efficiency in the creation of Ubiquitous games. By quickly making each of the diagrams types, it is easier for the end-user to validate if a desired application is feasible. This allows managing what are the required components and configurations at an appropriate level of abstraction in a domain expert using terminology of the domain instead of just computational terms.

References

1 Harold James Ruthven Murray: 'A History of Board-Games Other Than Chess' (Gardners Books, 1969. 1969)

2 Tiago Martins, Nuno Correia, Christa Sommerer, Laurent Mignonneau: 'Ubiquitous Gaming Interaction: Engaging Play Anywhere', in Heidelberg, S.B. (Ed.): 'The Art and Science of Interface and Interaction Design' (Vol.14, pag. - 115-130; Springer Berlin / Heidelberg, 2008)

3 http://www.hitachi.com/rd/sdl/glossary/u/ubiquitous_device.html, accessed 10th of July 2010

4 André Rosa: 'Designing a DSL solution for the domain of Augmented Reality Software'. Masters in Computer Sciences, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa, 2008/2009

5 Steven Kelly, Juha-Pekka Tolvanen: 'Domain-Specific Modeling: Enabling Full Code Generation' (John Wiley & Sons, Inc, 2008)

6 Tiago Martins, Teresa Romão, Christa Sommerer, Laurent Mignonneau, Nuno Correia: 'Towards an Interface for Untethered Ubiquitous Gaming'. Proc. 2008 International Conference on Advances in Computer Entertainement Technology, Yokohama, Japan

7 http://tiagomartins.wordpress.com/projects/noon-a-secret-told-by-objects/, accessed 10th of July 2010

8 Ronald T. Azuma: 'A Survey of Augmented Reality', Presence: Teleoperators and Virtual Environments, 4 August 1997, 6, pp. 355-385

9 https://launchpad.net/artoolkit/, accessed 10th of July 2010

10 http://ar.in.tum.de/Chair/ProjectDwarf, accessed 10th of July 2010

11 Prof. Bernd Bruegge Ph.D., Prof. Gudrun Klinker, Ph.D.: 'DWARF - Distributed Wearable Augmented Reality Framework'. Proc. Chair for Applied Software Engineering, Technische Universitat Munchen

12 Martin Bauer, Bernd Bruegge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Stefan Riß, Christian Sandor, Martin Wagner: 'Design of a Component–Based Augmented Reality Framework'. Proc. Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on, New York, USA 2001

13 Wesley M. Johnston, J. R. Paul Hanna and Richard J. Millar: 'Advances in dataflow programming languages'. Proc. ACM Computing Surveys (CSUR), New York, NY, USA March 2004

14 http://www.cycling74.com/products/max5, accessed 10th of July 2010
15 http://puredata.info/, accessed 10th of July 2010
16 http://processing.org/, accessed 10th of July 2010
17 http://www.openframeworks.cc/, accessed 10th of July 2010
18 Ricardo Guerreiro: ‘A DSML for Specification of Ubiquitous Games’. Masters in Computer Science, Faculdade de Ciênicas e Tecnologia, Universidade Nova de Lisboa, 2009
19 http://www.pranavmistry.com/projects/sixthsense/index.htm, accessed 10th of July 2010
20 http://www.blinkenlights.net/, accessed 10th of July 2010
21 Sara Ilstedt Hjelm: ‘Research + Design: the making of Brainball’. Proc. Interactions 2003 pp. Pages
22 Nuno Correia, Hélder Correia, Luís Alves, Luís Romero, Carmen Morgado, Luís Soares, José C. Cunha, Teresa Romão, A. Eduardo Dias, Joaquim A. Jorge: ‘InStory: A System for Mobile Access, Storytelling and Gaming Activities in Physical Spaces’. Proc. ACM SIGCHI - International Conference on Advances in Computer Entertainement Technology, Universidade Politécnica de Valência, Valência, Spain 2005
23 Staffan Bjork, Jennica Falk, Rebecca Hanson, Peter Ljungstrand: ‘Pirates! Using the Physical World as a Game Board’. Proc. Interact 2001, Tokyo, Japan 2001
24 Steve Benford, Martin Flintman, Adam Drozd, Rob Anastasi, Duncan Rowland, Nick Tandavanitj, Matt Adams, Ju Row-Far, Amanda Oldroyd, Jon Sutton: ‘Uncle Roy All Around You: Implicating the City in a Location-Based Performance’. Proc. International Conference on Advances in Computer Entertainement Technology (ACE) 2004, Singapore
25 Irma Lindt, Jan Ohlenburg, Uta Pankoke-Babatz, Wolfgang Prinz, Sabiha Ghellal: ‘Combining Multiple Gaming Interfaces in Epidemic Menace’. Proc. Conferences on Human Factors in Computing Systems 2006, Montréal, Québec, Canada
26 http://www.headbanghero.com/, accessed 10th of July 2010
27 Vasco Sousa: ‘Model Driven Development Implementation of a Control Systems User Interfaces Specification Tool’. Masters in Computer Sciences, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa, 2008/09
28 Arie Van Deursen, Paul Klint, Joost Viser: ‘Domain-Specific Language: A Annotated Bibliography’: ‘ACM SIGPLAN NOTICES’ (Vol. 35, Issue 6, pag. 26-36; ACM, New York, USA, 2000)
29 Vasco Sousa, Vasco Amaral and Patrícia Conde: ‘Towards a full implementation of a robust solution of a Domain Specific Visual Query Language for HEP Physics analysis’. Proc. Computing in High Energy and Nuclear Physics (CHEP) 2007
30 http://www.uml.org/, accessed 10th of July 2010
31 Krzysztof Czarnecki: ‘Overview of Generative Software Development’. Proc. Unconventional Programming Paradigms (UPP) 2004, Mont Saint Michel, France
32 EMF: http://www.eclipse.org/modeling/emf/; GMF: http://www.eclipse.org/modeling/gmf/, accessed 10th of July 2010
33 Jakob Nielsen and Thomas K. Landauer: ‘A mathematical model of the finding of usability problems’. Proc. ACM INTERCHI'93 Conference, Amsterdam, Netherlands, April 1993
34 Pedro Gabriel: ‘Software Languages Engineering: Experimental Evaluation’, Faculdade de Ciênicas e Tecnologia, Universidade Nova de Lisboa, 2009
35 http://www.useit.com/alertbox/20000319.html, accessed 10th of July 2010