The RPG DSL: a case study of language engineering using MDD for Generating RPG Games for Mobile Phones

Eduardo Marques
e.marques@campus.fct.unl.pt

Valter Balegas
balegas@live.com

Bruno Barroca
mailbrunob@gmail.com

Ankica Barišić
akki55@gmail.com

Vasco Amaral
vasco.amaral@gmail.com

Departamento de Informatica
Faculdade de Ciencias e Tecnologia
Universidade Nova de Lisboa, Portugal *

ABSTRACT

It is typical in the domain of digital games to have many development problems due to its increasing complexity. Those difficulties include: *i)* little code reuse in order to develop a cross-platform game; and *ii)* performing game's verification through extensive and expensive tests. This of course results in low productivity in the development (evolution and maintenance) of game solutions.

In this paper, we present a domain-specific language (DSL) for a Role-Playing Game (RPG) product lines, which was completely built using a software development technique driven by high level abstractions—called Model-Driven Development (MDD). Also, we discuss and demonstrate the several benefits of applying MDD in terms of rapid prototyping of cross-platform games, and their evaluation by means of static and dynamic verification of the game's logic properties.

Categories and Subject Descriptors

H.1.0 [**Information Systems Applications**]: Models and Principles; D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Model-Driven Development, Domain Specific Language, Model transformation, Algebraic Petri-net

Keywords

Model-Driven Development, Domain-Specific Language, Model transformation, Algebraic Petri-net, Role Playing Games, Game Analysis

1. INTRODUCTION

The increasing complexity of software development– mostly due to the increasing complexity of the Functional and Non-Functional requirements involved (problem domain), and the supporting platforms and technology (solution domain)– has been the main challenge of software engineering as research topic since its origins. The lack of reuse for the new solutions [3,6] and the lack of infrastructures that allow rapid development to improve the development life-cycle in what concerns to requirements' validation, have been some of the main reasons and consequences of the poor quality of Software Projects. Software game development is no exception: the process of game development is usually associated with low productivity—it may take years to see the final product. This is due the fact that the developed games have complex graphics, logic, artificial intelligence and input devices [2]. Their validation is performed through extensive and costly tests, and most of them are cross-platform. This last characteristic means however that there is a potential gain in reusing the produced software game not only to reduce development costs, but also to reduce verification's and validation's costs.

In the Game Domain, we observed that games can be organized into a wide range of categories each one sharing common game logic. For instance, in the category of Role-Playing Games (*RPGs*), all games tend to share the same concepts (e.g., characters, dialogues, maps and quests), regardless of the underlying implementation technology. Again, this characteristic is a potential target for code reuse at the game logic level. Therefore it makes sense to build a Domain Specific Modeling Language (*DSML*) to design, and deploy RPG games.

The work of software language engineers [10] is to develop languages that are able to provide those abstractions to the Experts in a given Application Domain — or in our case a Game Designer. These languages must be simple, focus on the domain of the problem, and use a vocabulary that is natural to the domain expert. As such, software language engineers can use MDD techniques to develop languages by having models of both the language's syntax (by means of Meta-models), and the language's semantics (by means of transformation models).

In this paper, we present an example of SLE (Software Language Engineering) while applying MDD techniques, where we engineer a visual DSML for a Role-Playing Game (*RPG*) product line (we can create different RPG games using the

*The presented work has been developed in the context of the following research institution: CITI fund PEst-OE/EEI/UI0527/2011 Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) - 2011-2012, and the doctoral grant ref. SFRH/BD/38123/2007.

same DSML). The end products (each individual game) is deployed in a smartphone platform. With this language, the Game Designer is able to model a game in terms of rules, challenges, characters, etc., and generate the code for a given target platform. It is possible to build a series of RPG games with diverse features, creating this way different types of games: one may have only mazes to solve; another may have agents to interact; etc. In addition, we can perform analysis by using Model Checkers [1], verifying properties on the designed games such as: it is possible to finish the specified game; or that it is possible for a player to get the maximum score (in w.r.t. the score definition in the specified RPG game).

2. THE APPROACH

In this section we discuss the tools used to implement our solution [1] and show a complete workflow to generate RPG games and verify them. This methodology is reusable and can be applied to any type of MDD project. The process is divided in three stages: Domain Analysis, Design, and Implementation. Briefly, in the Domain Analysis phase we define the domain of the application being developed and the domain for the target platform, on which we want to deploy the solution. In this phase, it is also important to develop a vocabulary that is easy for the domain experts to use. In the Design phase, we precisely describe the previously analysed domains. These descriptions (or models) are the input for the Implementation phase. Finally, in the Implementation phase, we realize (by means of transformation models) the models resulting from the Design phase into concrete artifacts in a target platform—let it be an execution or an analysis platform, or in our particular case: both.

2.1 Domain Analysis

In the Domain Analysis phase we worked in two different levels, at the level of the problem (i.e., expressing the concepts and logics of RPG Games), and at the level of the solution (i.e., how those concepts can be realized in a computational platform). In the level of the problem of RPGs' design, we tried to express and define what would be the common characteristics of all RPG games, regardless of what are the requirements of their implementation on an underlying computation infrastructure. In the following subsections, we describe the domain that we analysed for the RPGs. From this analysis, we define our DSML's syntax by means of a Meta-model.

2.1.1 Concept Agents

Agents are the characters of an RPG that occupy some cell in a scene, and with whom the hero may interact. The agents have attributes, inventory, resources and a set of actions. There is a broad variety of attributes such as strength, agility, intelligence, health points (these are mandatory on every game) or magic points. The health points may be recovered using items, or recovered with time, but if they reach 0, the agent dies. Also, an agent has an inventory where all the items, resources and equipment, gathered by him/her, are kept. Finally, the agents' actions can be specified using our RPG language. For instance, in dialogues, the phrases said by the agent are determined with a decision tree, but, during a combat, the selected movement is randomly selected in a set of possible fighting movements. The possible actions of agent are to walk, to fight, to talk and to give, buy or sell items. When an agent dies it automatically disappears from the map, and it may leave behind some items in its place, or give some resources to the hero.

2.1.2 Concept Hero

The hero is the controllable character of any RPG — i.e., it is the agent that the human player controls with a broader set of actions. When the hero dies, the game ends. The possible set of actions available to a hero, in our RPG game language, are: to move between cells, to interact with other agents by talking or fighting them, to interact with items and objects by picking them up, giving, buying or even selling them. Resources, such as gold, may be used to buy other items, these are gained throughout the scenes or by fighting hostile agents. All of the specified RPG benefit from an experience system, where a hero's abilities improve through the accomplishment of objectives and interaction with other agents or object. A hero can gather a small amount of experience points by accomplishing those tasks and experience points can be spent to improve the hero's attributes. Also, it is possible to equip some items of the inventory, that can improve some of the attributes of the hero, as long as they are equipped. The inventory can be checked or modified at any time. The hero attributes can also be modified by state conditions (e.g., poisoned, burned or sleepy).

2.1.3 Concept Space

Every game has a world map, which is the environment where the game takes place. The world map is composed by many different scenes, which are connected, and the agents can move across. A scene contains a two-dimensional map of cells. The agents can move between cells if they are unoccupied. In the map there exist objects or artifacts to be picked up, traps that cause the agents to lose health points, switches that, when activated, let the hero progress to another scene, and doors that allow the passage of agents to other scenes.

2.1.4 Concept Objectives

Each game has a main objective, that once finished ends the game, and there may also be other objectives, which are considered secondary objectives. There are different kinds of objectives: interaction with agents (e.g., talk with a specific agent), get to a specific scene (e.g., arrive at castle), or get artifact (e.g., get golden cup). The number of objectives completed determines the final score of the game. In Figure 1, we show a piece of the feature model for the RPG language. The feature model expresses the features that are mandatory or optional and also the relations between them. We used the feature model notation to represent the variability of RPG games schematically. Here we can see how we modelled the space elements defined in the Domain Analysis phase.

With this complete feature model for RPG games we derived our RPG DSL's syntax (by means of an Ecore Meta-model) based on the relations discussed above.

[1]For more details, our solution is available online at http://solar.di.fct.unl.pt/twiki5/pub/Projects/BATIC3S/ReleaseFiles/RPGCaseStudy.zip

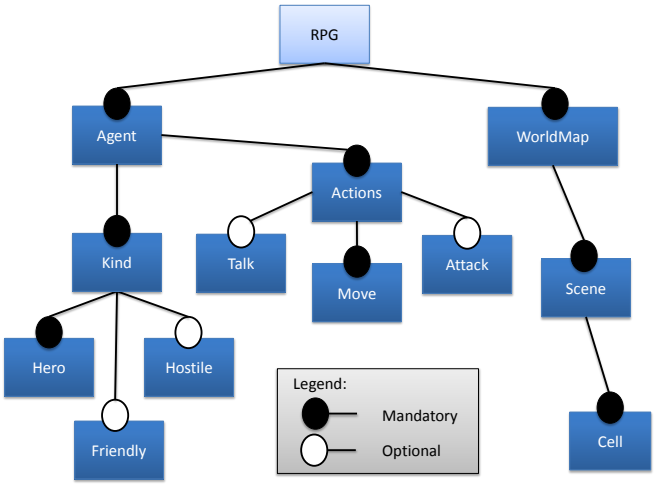

Figure 1: Excerpt of the RPG Games feature model.

2.1.5 *Execution Platform*

In the Domain Analysis of the solution (computational) level, we had to choose platforms to both deploy the generated games and to analyse them. Since we did not have any kind of experience in this area, nor any base prototype to work on, we had to choose a game developing platform and build our own framework on top of it. So we analysed some of the existing game engines that could be useful to implement our RPG Games. The criteria to choose the target framework were: fast development; abstraction level relatively to system calls and hardware dependencies (e.g., graphical primitives, input modalities, etc.); need of previous knowledge in the area of game engines.

We analysed three frameworks and identified some of its characteristics that we considered advantageous in the use of each framework. In the table 1 we describe these.

Table 1: Comparative table between frameworks

Framework	characteristics
Slick	Java based; Uses LWJGL
Sphere	Scripting language; Level of abstraction that allows some of the typical features of RPGs
Corona	Scripting language; Allows cross-platform compilation for Android and iOS

Between these three frameworks we chose Corona SDK [2] because it seemed interesting to allow compiling the game for different mobile platforms. The game development is done in the Lua language: a scripting language, which is preferred for rapid development [19].

2.1.6 *Analysis Platform*

With regard to the choice of the analysis platform, we had first to analyse what are the analysis requirements for our RPG game specifications. As in every game development it is possible to create games that are impossible to finish, so it is expected that the games automatically generated in our approach have models in the DSL that comply to some desired properties. We used OCL [8] to perform static analysis over our RPG models. This kind of verification assures that models are well formed and therefore can be used through the MDD cycle. However, they cannot guarantee that dynamic properties are valid over all possible computation steps (also known as symbolic states) of the game. In particular, in our RPG Game Language, we want to assure that all developed games have the possibility of *i)* finishing a game, and *ii)* finish a game with the maximum score.

To check this kind of properties we integrated a model checker [1] in our DSML. There are various model checkers in the literature [9, 12, 13], but we choose ALPiNA [4], an Algebraic Petri-net (APN) [11] analyser. Although this tool suffers (as other model checkers) from the exponential nature of the model checking problem, where the analysis space tends to explode with the size of the problem [5], this tool actually presents good performance while checking invariants in Petri net models. Also its APN models are expressed with the same ECore format as the models we use to define new RPG games, therefore, we could integrate the code verification in the MDD cycle seamlessly using this tool, by performing the transformation from our RPG model's to APN model's and by analyzing the latter ones with ALPiNA.

2.2 Design

After making an overview of the requirements of our language we defined the required Meta-models to implement it. We used Ecore-based Meta-models used by the Eclipse Modeling Framework (EMF) [17] [3]. The Meta-models describe the RPG Language and the abstractions of the target platforms. This is the base of the whole process of deploying the RPGs. A bad design of Meta-models may lead to severe changes in its implementation, execution, analysis and graphical editors. In our project we created three Meta-models, one for the main language from which we are able to design RPG games, and two other intermediate languages to simplify the process of transforming these instances to both the execution and analysis platforms.

In Figure 2, we can see the automated transformation specifications that are used to translate RPG game specifications into APNs, and to the code that is used in the Corona Framework. Instead of performing direct Model-to-Model(M2M) either from RPG models to APN, or Model-to-Code(M2C) from RPG models to the framework code, we decided to add intermediate steps to this process. The main reasons to introduce these intermediate steps are: *i)* to enable further reuse of the model transformation when approaching other target platforms (for both execution and analysis); *ii)* to enhance debugging capabilities by inspection on the results of the intermediate transformations; and *iii)* to ease and structure the implementation of the RPG language, and ease future language evolutions.

[2] http://www.anscamobile.com/corona/

[3] http://eclipse.org/modeling/emf/

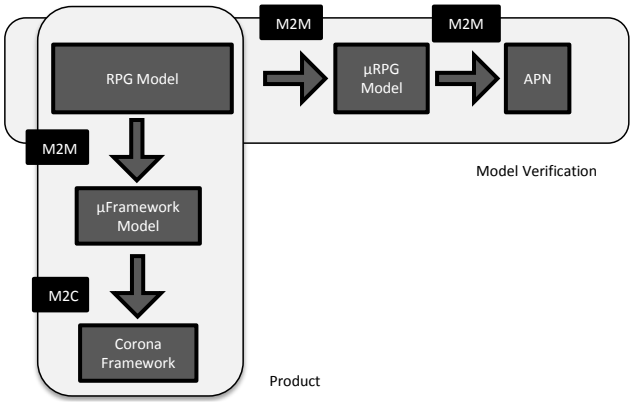

Figure 2: Transforming a source model to a target framework with model verification.

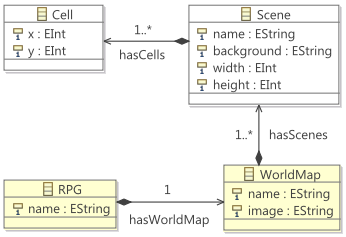

Figure 3: Excerpt of the RPG Meta-model.

2.2.1 RPG Language Meta-model

The RPG Language Meta-model describes every possible feature of the defined RPG Domain. We used the feature model, created in the previous section, to design this Meta-model. It was annotated with OCL rules to guarantee that every produced RPG model is consistent by verifying rules such as "there can be only one hero in the world". This is crucial since both the execution and analysis transformations are assuming that the source RPG models are always correct.

In Figure 3, we show the part of the Meta-model that describes the Space entities. In this example, a RPG Game has only one World Map, that has a set of Scenes, which are composed by Cells. Cells are identified by x and y coordinates, a Scene is identified by its name, has an image for the background and its number of Cells is delimited by width and height. With this model of our RPG language, we generated a graphical editor to allow the Game Designers to create RPG instances with it, as described in section 2.3.1. The created instances are then transformed into the other two languages: the Corona's Framework code, and the APNs used in ALPiNA.

2.2.2 μFramework Meta-model

The application domain of the chosen target framework (Corona) is very broad and general—thus it had only low-level primitives to draw and create graphical objects. Therefore, we built an API over it, in order to create the entities we needed to our RPG games and created a Meta-model for this API. This was called the μFramework API, which simplifies the

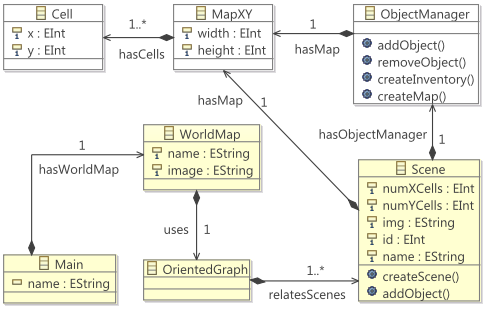

Figure 4: Excerpt of the μFramework Meta-model.

M2C transformation to generate the code of the game. This way, we restricted the target framework to the RPG game context. The created Meta-model simply describes the functionality implemented in the μFramework API, mapping the entities to the functionality of the API.

In Figure 4, we can see that similarly to the RPG meta-model, this μFramework Meta-model has only one *WorldMap*, but now also uses a *OrientedGraph* that relates and store all the *Scenes* that the specified game may have. Each *Scene* has one MapXY, which holds all of the Cells of that Scene, and one *ObjectManager* that is responsible for managing the objects in the scene (e.g., placing an object, remove an object, etc.).

2.2.3 μRPG Meta-model

Given the analysis limitations of the ALPiNA model checker, we built another intermediate language (μRPG Language) which works as a filter that will only contains the entities we considered essential to check the properties related to the termination and score of a game. In this process we made several assumptions to reduce the number of entities to a minimum. As for example of one of these assumptions is that a hero can always beat an enemy, this has a huge impact since all enemies will be discarted from the process of model checking, which can lead to false positives where there may exist overpowered enemies that will block our way to the final goal.

The μRPG Meta-model one of modifications that occured in this meta-model is the concept of Partition which represents sets of adjacent cells that can be directly accessed by a hero: a given Partition holds the relevant information of that area w.r.t. the property of game termination—i.e., the objectives, the keys that can be picked up, the doors, and the hero position.

2.3 Implementation

In this section, we describe how we can create RPG models and use those models to automatically generate both the game source code and the models for verification.

2.3.1 Graphical Language

We developed a graphical language based on GMF/EuGENia[4] that allows the creation of RPG models. The original RPG

[4]http://www.eclipse.org/gmt/epsilon/doc/eugenia/

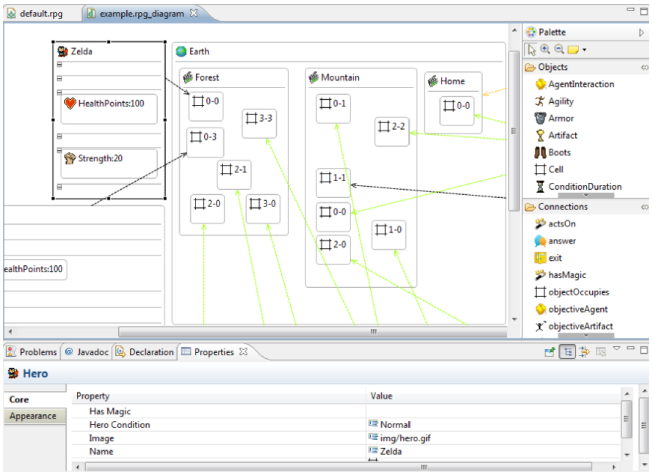

Figure 5: Screenshot of the graphical editor.

Language Meta-model was annotated with rules that describe how the entities and relations are represented in the GMF Editor.

In the Figure 5, we show an example of an RPG game created in this language. In the main window, the entities of the RPG model and the relations between them. In the properties window bellow, we can assign values to the RPG entities' attributes: in this case, we selected the Hero entity named 'Zelda'.

2.3.2 *Generating the source code*

The generation of source code for the RPG Game is divided in two transformation phases: First, we take a RPG Game instance and transform it to the μFramework Meta-model instance, using a M2M transformation. Then, we take the μFramework Meta-model instance and apply a M2C transformation, as can be seen in Figure 2.

All of the M2M transformations were done by means of the Atlas Transformation Language (ATL) [5], which is a textual model transformation language that is able to perform M2M transformations. The M2C transformation was specified using XPand[6], which is a template based tool to generate the source code of programs. Template based tools generate the source by reusing snippets of code. Those snippets are filled with values from the model instances. The information from the Meta-model is read in a visitor-pattern style, and it is used to generate the appropriate textual code. This technique is appropriate for cases where we already have a considerable amount of code from the target platform, and we want to reuse it in the MDD cycle.

2.3.3 *Generating μRPG*

To generate the μRPG Language, we made a simple M2M transformation that just propagates the relevant the entities from the RPG model to the model expressed in the μRPG Language. We implemented a breadth search algorithm to translate reachable adjacent cells in a scene into a Partition.

[5]www.eclipse.org/atl/
[6]http://wiki.eclipse.org/Xpand

We then take the generated RPG model expressed in the μRPG language, and translate it again into an APN model to be analyzed in ALPiNA. This M2M transformation generates an APN from an μRPG model, where: *i)* each partition is mapped into a place; *ii)* Keys, objectives and the hero are mapped to tokens; *iii)* doors are mapped to transitions. In the translated APN there will always be two extra places: the KeySet that holds the keys picked up by the hero; and the Conquests that holds finished objectives. The transition associated with the door, allows the token associated with the hero to move from place to place, and will be enabled if and only if there is a token associated with the respective key in the KeySet place. We can put this token in the KeySet whenever the token associated with the hero is in the place that has that token. This also is applied to objectives.

Finally, to check if the game ends, we use AlPiNa to verify that there is a state which has the token associated with the final objective in the place Conquests. For the maximum score property, where we check if the cardinality of place Conquests can eventually be equal to the total number of objectives defined in the game.

3. RELATED WORK

MDD is often applied to deliver a good separation of the concepts of the application from the concepts of the system. This separation improves productivity and communication because teams concerned with the domain of the application can easily talk with system developers without concerning the algorithmic requirements.

Besides the BATIC3S [16] project, there exists not so many evidence of successful application of MDD solutions in the SLE of a DSML. Nevertheless, related MDD approaches already have been introduced to the Game Industry before. In [15] it is proposed to adopt MDD in the Game Industry by proposing a modelling semi-automatic approach based on UML (that can be seen as a General Purpose Modelling language for Software Engineering) at several abstraction layers: Platform-Independent Models (PIM), Platform-Specific Models (PSM) and code level. However not demonstrated, the authors claim increasing productivity and higher code re-utilization, not mentioning how it might affect the error proneness caused by the fact the game developer must model and code, and keep track of consistency, in all the three layers. Code generation, in this case is not complete and can be seen more like a code skeleton generator.

There are several DSMLs built for game development. For instance, in [7] and [18] it was presented DSMLs for Adventure Games, with automatic code generation for a specific adventure game. However, our solution provides model checking capabilities (reducing costs with exhaustive game testing) and an abstract representation of the deployment platforms (tackling the problem of platform heterogeneity).

4. CONCLUSIONS AND FUTURE WORK

In this work, we developed a DSML to create RPG Games with a complete MDD approach. This includes the M2M transformations and intermediate languages to tackle the complexity of building a DSML and the use of a model checker to verify some game properties.

The creation of an API over the framework allowed an M2M transformation that was easily produced between RPG and μFramework meta-models, which consists mostly of 1 to 1 relationships. This strategy of bottom-up modelling in the framework allowed the focusing on the RPG entities and the restriction of the power of the framework which was very low-level. The mapping of the RPG meta-model to APN is too much complex to simply perform it in just one step, therefore we created an intermediate meta-model μRPG that allowed a simpler mapping between both. We believe that the use of intermediate languages really help in this process, since we do not have to map complex entities directly to an APN. To complete the MDD cycle, the use of ALPiNA demonstrates that we can use a model checker to analyse and validate properties on RPG games, giving us a certain level of confidence about its implementation, since it passed verification phase.

Regarding the RPG metamodel evolution, the addition of a new feature should not affect the existing ones if its concept does not interfere with existing features. However if it does interfere, we have to analyze the impact of that interference in the μRPG Meta-model, which may lead to a partial redefinition of these model.

The choose of the DSL format (textual or graphical) has impact in the process of game development. A textual DSL may lead to a more readable solution than a graphical one in the development of big games, since a visual one will have problems displaying all the information about the game. However a graphical DSL allows the developer to get a preview how the things will be mapped, allowing the game developer detect errors faster.

As future work we are interested in collecting some metrics and conducting a complete assessment to gather the opinions from both game engine developers and game designers about their experience with this, or similar MDD approach. We are interested in comparing the productivity of game designers using this methodology and others. In a different survey, we will investigate the difficulty of developing a game language to generate games, in a MDD fashion, against the development of a generalist game engine, as the ones broadly used in the industry. Either for small game devices, such as the ones used in smartphones, or other more complex environments.

5. REFERENCES

[1] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer Verlag, 1st edition, 1999.

[2] J. Blow. Game development: Harder than you think. *Queue*, 1:28–37, February 2004.

[3] B. Boehm. Managing software productivity and reuse. *Computer*, 32(9):111 –113, sep 1999.

[4] D. Buchs, S. Hostettler, A. Marechal, and M. Risoldi. Alpina: A symbolic model checker. In *Petri Nets*, pages 287–296, 2010.

[5] D. Buchs, S. Hostettler, A. Marechal, and M. Risoldi. Alpina: An algebraic petri net analyzer. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 349–352. Springer, 2010.

[6] G. Caldiera and V. Basili. Identifying and qualifying reusable software components. *Computer*, 24(2):61 –70, feb 1991.

[7] A. W. B. Furtado and A. L. M. Santos. Using domain-specific modeling towards computer games development industrialization. In *Domain-Specific Modeling workshop at OOPSLA 2006*, 2006.

[8] O. M. Group. *Object Constraint Language OMG Available Specification Version 2.0*, 2006.

[9] G. Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, first edition, 2003.

[10] A. Kleppe. The field of software language engineering. In *SLE*, pages 1–7, 2008.

[11] C. Lakos. From coloured petri nets to object petri nets proceedings of 15th international conference on the application and theory of petri nets. 1995.

[12] M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialisation. In A. Bossi, editor, *Logic-Based Program Synthesis and Transformation. Proceedings of LOPSTR'99, LNCS 1817*, LNCS 1817, pages 63–82. Springer-Verlag, Berlin, September 1999.

[13] M. Leuschel and T. Massart. Logic programming and partial deduction for the verification of reactive systems: An experimental evaluation. Technical report, University of Birmingham, 2002.

[14] R. Prieto-Diaz. Status report: software reusability. *Software, IEEE*, 10(3):61 –66, may 1993.

[15] E. M. Reyno and J. . C. Cubel. Automatic prototyping in model-driven game development. *Computers in Entertainment*, 7(2), 2009.

[16] M. Risoldi, V. Amaral, B. Barroca, K. Bazargan, D. Buchs, F. Cretton, G. Falquet, A. L. Calvé, S. Malandain, and P. Zoss. A language and a methodology for prototyping user interfaces for control systems. In *Human Machine Interaction*, pages 221–248. 2009.

[17] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[18] R. Walter and M. Masuch. How to integrate domain-specific languages into the game development process. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ACE '11, pages 42:1–42:8, New York, NY, USA, 2011. ACM.

[19] W. White, C. Koch, J. Gehrke, and A. Demers. Better scripts, better games. *Queue*, 6:18–25, November 2008.