J. Software Engineering & Applications, **2010, 3, 1040-1046**
doi:10.4236/jsea.2010.311122 Published Online November 2010 (http://www.SciRP.org/journal/jsea)

Generation Rules in POMA Architecture

Mohamed Taleb[1], Ahmed Seffah[2], Alain Abran[1]

[1]École de technologie supérieure (ÉTS), Montreal, Canada; [2]Concordia University, Montreal, Canada.
Email: mohamed.taleb.1@ens.etsmtl.ca, seffah@cse.concordia.ca, alain.abran@etsmtl.ca

Received August 21st, 2010; revised September 14th, 2010; accepted September 20th, 2010.

ABSTRACT

Another component in Pattern-Oriented and Model-Driven Architecture (POMA) is the concept of model generation. The generation code of models is the process of creating a source code from a model using generation rules. In this paper, we present the generation rules that are used to support the automated code generator of POMA architecture to generate the source code of the entire interactive system. These Platform-Specific Model (PSM) models are based on patterns which illustrate how several individual models of patterns can be generated at different levels of abstraction such as PSM models to source code in the development of interactive systems.

Keywords: *Patterns, Models, Architecture, Interactive Systems, POMA, Generation Rules*

1. Introduction

Over the past two decades, research on interactive systems and user interfaces (UI) engineering has resulted in several architectural models which constitute a major contribution not only to facilitate the development and maintenance of interactive systems, but also to promote the standardization, portability and ergonomic "usability" (ease of use) of the interactive systems being developed. POMA architecture [1] is developed to take into account these software quality factors.

Several interactive system architectural models have been introduced. Buschmann *et al.* define architectural models as [2]: "*the structure of the subsystems and components of a system and the relationships between them typically represented in different views to show the relevant functional and non functional properties*." This definition introduces the main architectural components (for instance, subsystems, components, and connectors) and covers the ways in which to represent them, including both functional and non-functional requirements, by means of a set of views.

Software architectures defined in terms of target language patterns, design rules and implementation technologies are incorporated into a translator that generates code for the target system. The software architectures are completely independent of the applications they support [3].

The software development in Model-Driven approach is popular due to the ubiquity of the code generation techniques and languages which constitute a major challenge to software engineering.

Code generation is an extremely valuable tool that can have a stunning impact on productivity and quality in software engineering projects. Code generation is the technique of writing and using programs that build application and system code. Herrington [4] proposes a set of top ten code generation rules used for designing, developing, deploying, and maintaining the code generator. These rules are used specifically to build code generators.

Automated code generation is not a new concept, although it has seen relatively little recognition until recently. Recent changes in the way software is developed may cause an upsurge in the use of code generators as a way of bringing enterprise software to market extremely quickly [5].

A number of code generation languages and tools have been proposed. For example, XDoclet for Java using Code-Driven Approach. XSLT, Velocity, and Jostraca show how to build textual output from an input specification and the code by specifying a model of the code as input, using the template to specify the code using Model-Driven Approach called Custom approach. OptimalJ, AndroMDA, MDE, ArcStyler show how to generate the code from PSM model using Model-Driven approach called Model-Driven Architecture [4]. The customization approach, actually, gives the power to apply code generation techniques to new areas of the project, as and when they are identified [5]. JGenerator "bootstrap" tool shows how to generate the first pass of this file from the database meta-data, to give the programmer a head-start tool using a properties file or XML file approach to contain the additional meta-data and processing directives. JavaDoc tool shows how to generate code from a heavily annotated skeleton class [5].

Matlab/Simulink [6] focuses on data visualization, algorithms, analysis and numeric computing. The code can be automatically generated from models. Giotto [6] is a time-triggered language for embedded control system which is developed by the University of Berkeley. It supports the automation of control system designed by strictly separating platform-dependent functionality from scheduling and communication. Currently, unified modeling language (UML) [6] is the most popular modeling language. Although some diagrams are suitable for automatic code generation, the implementation must be done by hand. As a general purpose modeling language, UML is unable to describe embedded control system characteristics such as deadline, and fault-tolerance.

Patterns have been proposed to alleviate some of these weaknesses, and indeed were introduced based on the observation given by Alexander [7]. Such a pattern provides, on a single level, a pool of proven solutions to many of the recurring weaknesses. Patterns have proven their utility in different fields of applications.

In 2001, the Object Management Group introduced the Model-Driven Architecture (MDA) initiative as an architecture to interactive system specification and interoperability based on the use of formal models (*i.e.*, defined and formalized models) [8].

In recent years, interactive systems have matured from offering simple interface functionality to providing intricate processes such as end-to-end financial transactions. Users have been given more sophisticated techniques to interact with available services and information using different types of computers. Different kinds of computers and devices (including, but not limited to, traditional office desktops, laptops, palmtops, Personal Digital Assistants (PDAs) with and without keyboards, mobile telephones, and interactive televisions) are used for interacting with such systems. One of the major characteristics of such cross-platform interactive systems is that they allow a user to interact with the server-side services and contents in various ways. Interactive systems for small and mobile devices are resource constrained and cannot support a full range of interactive system features and interactivity because of the lack of screen space or low bandwidth. One important question is how to develop and deploy the same system for different platforms - without "architecturing" and specifically writing code for each platform, for learning different languages and the many interactive systems design guidelines that are available for each platform.

In our continued research project, the goal can be stated as follows: "Define a new architecture to facilitate the development and migration of interactive systems while improving their usability and quality." To pursue this goal, the research objective is to define the generation rules for POMA architecture [1]. This generation will cover the different levels of abstractions of POMA architecture such as Domain, Task, Dialog, Presentation and Layout of Platform-Specific Model (PSM) models. The **Figure 2** summarizes the model generation rules that were defined and applied to PSM models represented in **Figure 1**.

2. Background and Related Work

The main idea of MDA is to specify business logic in the form of abstract models. These models are then generated (partly automatically) according to a set of code generation rules to different platforms. The PSM models are usually described by UML in a formalized manner which can be used as inputs for tools which perform the generation process. The main benefit of MDA is the clear separation of the fundamental logic behind a specification from the specifics of the particular middleware that implements it. The MDA approach distinguishes between the specifications of the operation of a system and the details of the way that the system uses the capabilities of its platform.

POMA architecture for interactive systems engineering [1] identifies an extensive list of pattern categories and types of models aimed at providing a pool of proven solutions to these problems. The models of patterns span several levels of abstraction, such as domain, task, dialog, presentation and layout. The proposed POMA architecture illustrates how several individual models can be combined at different levels of abstraction into heterogeneous structures which can then be used as building blocks in the development of interactive systems.

Subsequently, the different components of POMA architecture were detailed in [1] including:

- The architectural levels and different categories of patterns [9,10] and [11];
- The Platform-Independent Model (PIM) and PSM models [12];
- The pattern composition rules to select and compose patterns corresponding to each type of PIM model [9] and [11];

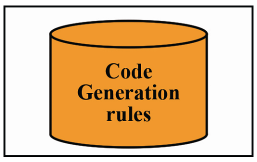

Figure 1. Generation rule in POMA architecture.

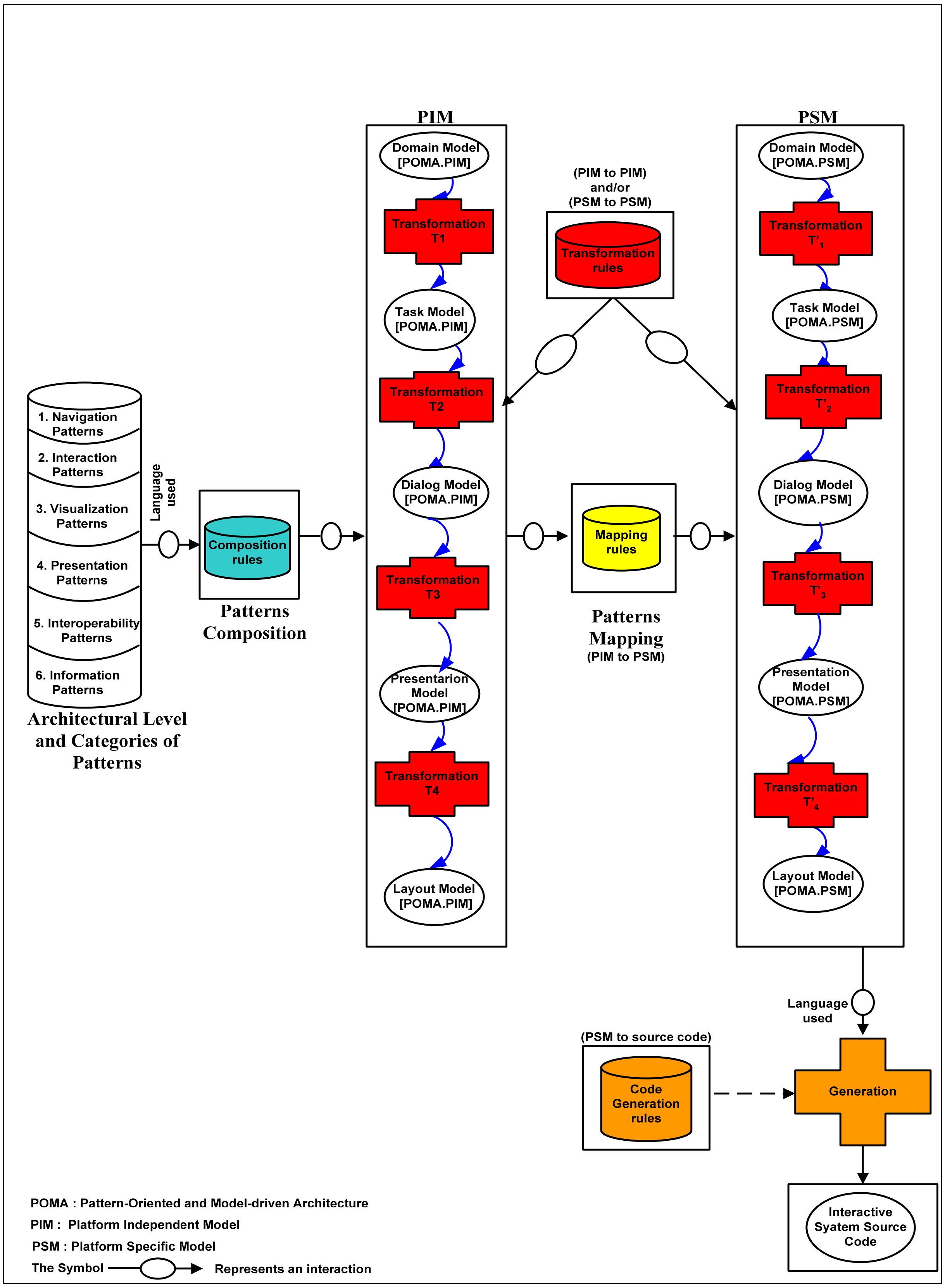

Figure 2. POMA architecture [1].

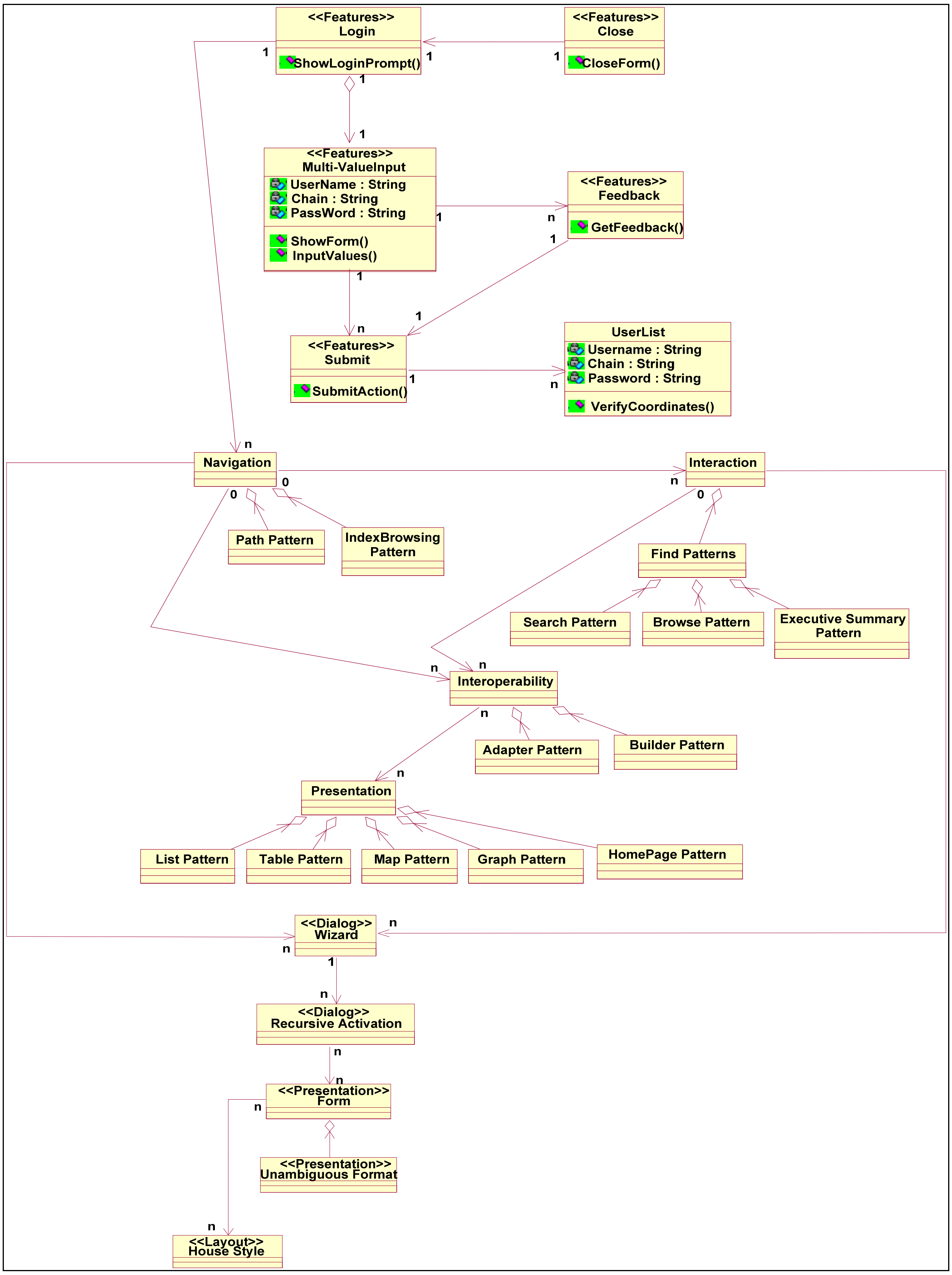

Figure 3. UML class diagram of a PSM model.

Table 1. Generation rules for the PSM model for a laptop platform.

Step (1)	Object type of a Layout model (2)	Object Name (3)	Generation Rule (4)	After Generation for Laptop platform (5)	Comments (6)
❶	Class	'Login'	Analyser	- Analysing successfully. - Otherwise return to its design.	Analyser rule: - Detects all properties of the 'Login' class such as Domain, Domain service, Attributes, Class service, State, Event, Transition, Superstate, Substate, its methods. - Detects also the relationship of others classes such as Association, Inheritance, Associative class. - Ensures and avoids design errors and misspellings of 'Login' class.
❷	Class	'Login'	Interpreter	- Interpreting successfully. - Otherwise return to its analysis (Step❶).	Interpreter rule : - Converts the 'Login' class in XML code. - Sends the XML code to the 'Parser" rule for compiling.
❸	Class	'Login'	Parser	- Parsing successfully. - Otherwise return to its Interpretation (Step❷).	Parser rule: - Verifying and compiling the XML code about the existence and the syntax of 'Login' class if is conformed.
❹	Class	'Login'	Transmitter	Transmitting successfully. Note: This step ❹ will be applied in the final process, *i.e.*, after the steps❶, ❷ and ❸ are executed for all PSM model in order to generate the entire application.	Transmitter rule: - Establishes the relationship between 'Login' class and automated Code Generator of POMA architecture. - Transmits all object properties of 'Login' class automated Code Generator for generating the entire of interactive system application.

- The pattern mapping rules to map the patterns and PIM models to produce PSM models for multiple platforms [9] and [11];
- The transformation rules for transforming PIM to PIM models and PSM to PSM models [13];
- The source code generation rules;
- The generation of the whole of application.

The strengths of POMA architecture include the following:

- POMA facilitates the use of patterns by beginners as well as experts;
- POMA supports the automation of both the pattern-driven and model-driven approaches to design;
- POMA supports the communication and reuse of individual expertise regarding good design practices;
- POMA can integrate all the various new technologies including, but not limited to, traditional office desktops, laptops, Palmtops, PDAs with or without keyboards, mobile telephones, and interactive televisions.

3. Generation Rules

To tackle some of the weaknesses identified in related work, we propose four generation rules for POMA architecture of pattern-oriented and model-driven generic classification schema for an interactive system. These rules will be specified, structured and described in Pattern-Oriented and Model-Driven Architecture Markup Language (POMAML) which is based on XML notation.

Generation is the description of a source code process from the five PSM models of POMA, *i.e.*, the process of converting the five PSM models – called source models – to an output source code – the target interactive system – of the same system. Generation may combine elements of different source models in order to build a target interactive system. Generation rules listed below apply to all the types of PSM models.

1) **Analyser**: This rule detects the PSM model objects such as Domain, Domain service, Class, Attribute, Association, Inheritance, Associative class, Class service, State, Event, Transition, Superstate, Substate, and avoids designing errors and misspellings in order to obtaining a clean, readable, optimized and error free by generating consistent, well-structured designing applications in interactive systems;
2) **Interpreter**: This rule takes the input PSM model, already analyzed by the '*Analyser*' rule, applies the model generation to perform language translation operations in XML code, and thus forwards it to the output '*Parser*' rule for verifying and compiling the generated XML code;
3) **Parser**: This rule provides a conversion for all in-

terpretation objects. The '*Parser*' must verify and ensure that no syntax error occurred and also the existence of the objects while processing performed by the '*Analyser*' and '*Interpreter*' rules before transmitting all the obtained and necessary properties to '*Transmitter*' rule in order to generate the source code of concerned PSM model objects by a code generator of POMA architecture.

4) **Transmitter**: This rule provides all PSM model objects and its properties such as classes and its contents, its relationship between classes, a chosen platform and language, etc. to code generator (automated tool) which allows developers to design, implement, and test their concepts in a platform-independent model of POMA architecture. This code generator converts the XML code of the PSM model into an interactive system source code in specific language for a specific platform which is included in the next research step.

4. Illustrative Case Study

This section describes the design illustrating and clarifying the core ideas of the POMA approach and its practical relevance. The interactive system and corresponding models will not be tailored to different platforms. This case study illustrates how generation rules are used to communicate and link the PSM model to code generator of POMA architecture.

This example presents a general overview of the class PSM model to the code generation of an interactive system by applying generation rules in POMA architecture for a specific platform. To do this, we will consider a single class which is '***Login***' of PSM model below to show the use of the generation rules in POMA architecture.

The **Figure 3** shows the [POMA.PSM] model which is obtained by transforming all models of patterns involved (Domain, Task, Dialog, Presentation and Layout) and applying the transformation rules [11].

Table 1 shows the generation rules for the PSM model for a laptop platform to the interactive system source code.

After the applied generation rules, the interactive system source code is obtained for a given platform.

5. Conclusion and Further Work

In this paper, novel practical generation rules for multi-platform POMA architecture are introduced for interactive systems engineering. Then, we proposed four generation rules of five PSM models to generation interactive system source code of POMA architecture. These rules allow preserving the integrity of PSM model design, well-strengthening and understanding the interpretation of the PSM models in a specific language and for a specific platform for clean, readable, optimized and error free design, maintaining the analysing structures of all PSM model (class instances and for association populations), and finally to supporting the automated code generator of POMA architecture.

Among the next steps required to develop POMA are:

- Development of a tool, *i.e.*, Code Generator, that automates the POMA architecture-based engineering process;
- Standardization of POMA architecture to all types of systems, not only to multi-platform interactive systems;
- Quality Assurance of the applications produced, since a pattern-oriented architecture will also have to permit the encapsulation of quality attributes and to facilitate prediction;
- Validation of the migration, the usability and overall quality of POMA architecture for interactive systems using different existing methods;
- Evaluation of the effectiveness and learning time of POMA architecture for novices and experts users.

REFRENCES

[1] M. Taleb A. Seffah and A. Abran, "Interactive Systems Engineering: A Pattern-Oriented and Model-Driven Architecture," *The* 2009 *International Conference on Software Engineering Research and Practice* (*SERP'*09) *in the* 2009 *World Congress in Computer Science, Computer Engineering and Applied Computing* (*WORLDCOMP*2009), Las Vegas, 2009.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, "Pattern-Oriented Software Architecture: A System of Patterns," John Wiley & Sons Ltd., Hoboken, 1996.

[3] Mentor Graphics, Internet available: http://www.mentor.com/products/sm/model_development/bridgepoint/upload/datasheet.pdf

[4] J. Herrington and Manning, "Generation Code in Action," Manning Publications Co., Greenwich, 2003.

[5] Matt Stephens, 2002. Internet available: http://www. softwarereality.com/programming/code_generation7.jsp

[6] G. H. Wu, D. W. Cheng and Z. Zhang, "A Solution Based on Modeling and Code Generation for Embedded Control System," *Journal of Software Engineering and Applications* (*JSEA*), October 2009, pp.160-164.

[7] C. Alexander, "The Timeless Way of Building," Oxford University Press, New York, 1979.

[8] D. D'Souza, "Model-Driven Architecture and Integration Opportunities and Challenges," OMG Group, 2001. Internet available: ftp://ftp.omg.org/pub/docs/ab/01-03-02. pdf

[9] M. Taleb, H. Javahery and A. Seffah, "Pattern-Oriented Design Composition and Mapping for Cross-Platform Web Applications," *The XIII International Workshop*

DSVIS 2006, Trinity College Dublin Ireland, Springer-Verlag, Berlin Heidelberg.

[10] M. Taleb, A. Seffah and A. Abran, "Pattern-Oriented Architecture for Web Applications," 3*rd International Conference on Web Information Systems and Technologies* (*WEBIST* 2007), Barcelona, 2007, pp. 117-121.

[11] M. Taleb, A. Seffah and A. Abran, "Patterns-Oriented Design for Cross-Platform Web-Based Information Systems," *The* 2007 *IEEE International Conference on Information Reuse and Integration* (*IEEE IRI*-07), Las Vegas, 2007, pp. 122-127.

[12] M. Taleb, A. Seffah and A. Abran, "Model-Driven Design Architecture for Web Applications," *The* 12*th International Conference on Human Centered Interaction International* (*FIC-HCII* 2007), Beijing, Vol. 4550, 2007, pp. 1198-1205.

[13] M. Taleb, A. Seffah and A. Abran, "Transformation Rules in POMA Architecture," *The* 2010 *International Conference on Software Engineering Research and Practice* (*SERP'*10) *in the* 2010 *World Congress in Computer Science, Computer Engineering and Applied Computing,* (*WORLDCOMP*2010), 2010, Las Vegas.