Transformation Rules in POMA Architecture

M. Taleb
Human-Centered Software Engineering, Group Concordia University, Montreal, Quebec, Canada
Telephone: +1 514 848 2424 ext 7166
mtaleb@encs.concordia.ca

A. Seffah
Human-Centered Software Engineering Group Concordia University Montreal, Quebec, Canada
Telephone: +1 514 848 2424 ext.3024
seffah@cse.concordia.ca

A. Abran
Software Engineering Department & Information Technology, École de technologie supérieure (ÉTS), Montreal, Quebec, Canada
Telephone: +1 514 396 8800 ext. 8632
alain.abran@etsmtl.ca

Abstract

Another component in Pattern-Oriented and Model-Driven Architecture (POMA) [1] is the concept of model transformation. The transformation of models is the process of creating a model from another model using transformation rules. In this paper, we present the transformation rules that enable to transform the five PIM models between them and also the five PSM models. These different PIM and PSM models are based on patterns which illustrate how several individual models of patterns can be transformed at different levels of abstraction such as PIM to PIM models and PSM to PSM models in the development of interactive systems.

Keywords: Patterns, Models, Architecture, Interactive Systems, POMA, Transformation 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.

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.

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

The crucial role of model transformation languages and tools for the overall success of model-driven systems development has been revealed in many surveys and papers during the recent years [3].

A number of model transformation languages and tools have been proposed. For example, VIATRA2 framework [3] which provides a rule and pattern-based transformation language for manipulating graph models by combining graph transformation and abstract state machines into a single specification paradigm, [4; 5] show how to transform a platform-independent model (PIM) into a more platform-specific model (PSM) using GReAT, [4; 6] illustrate the use of graph transformation for refinement of software architectures, [7] show how the transformation have been used for migration from one domain-specific language to another using GReAT, [8] illustrate the transformations by specifying formal refinements as typed graph transformations, [4; 9] explains how to transform a model refactorings in AGG and Fujaba languages.
Patterns have been proposed to alleviate some of these weaknesses, and indeed were introduced based on the observation given by Alexander [10]. 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 application.

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) [11].

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, 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 model transformation rules for Pattern-Oriented and Model-Driven Architecture (POMA) presented in [1]. This transformation will cover the different levels of abstractions of POMA architecture such as Domain, Task, Dialog, Presentation and Layout of PIM and PSM models. The Figure 1 summarizes the model transformation rules of POMA architecture that were defined and applied to PIM and PSM models represented by this symbol:

Figure 1: Pattern-Oriented and Model-Driven architecture [1]

2 Background work

The main idea of MDA is to specify business logic in the form of abstract models. These models are then mapped (partly automatically) according to a set of transformation rules to different platforms. The models are usually described by UML in a formalized manner which can be used as inputs for tools which perform the transformation 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 [12], [13] and [15].
- The PIM and PSM models [14].
- The pattern composition rules to select and compose patterns corresponding to each type of PIM model [12] and [15].
- The pattern mapping rules to map the patterns and PIM models to produce PSM models for multiple platforms [12] and [15].
- The transformation rules for transforming PIM to PIM models and PSM to PSM models.
- 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 Model Transformation Rules

To tackle some of the weaknesses identified in related work, set of concepts, proposes six transformation 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 based on XML notation.

Model transformation is the process of converting one or more models – called source models – to an output model – the target model – of the same system. Transformations may combine elements of different source models in order to build a target model. Transformation rules listed below apply to all the types of PIM and PSM models. ‘Controller’ and ‘Atomicity’ rules can be used and applied to the other rules during their treatment:

1. **Tracker**: This rule defines tracing levels for each transformation or for the entire model to maintain tracking structures of all class instances and for association populations. The first reason for tracking the dependencies between objects is to detect potential links for relating different models. The ‘Tracker’ rule requires some objects inputs.

2. **Sequencer**: This rule generates numeric values which are a references list for the model objects to enforce the events ordering and therefore a reference is attributed for each object.

   The ‘Sequencer’ is reusable and can be used in multiple transformation models such as PIM to PIM models and PSM to PSM models as shown in Figure 1. This rule can be used in large input models such as Domain, Task, Dialog, Presentation and Layout. The reusable sequencer can be used for these five models of POMA architecture [1] to provide a unique value for each target model.

   These values are used by the ‘Tracker’ rule to determine which object that can be linked to another object by the ‘Linker’ rule to create the link between these objects.

3. **Parser**: This rule provides a transformation for all analysis objects such as Domain, Domain service, Class, Attribute, Association, Inheritance, Associative class, Class service, State, Event, Transition, Superstate, Substate. The ‘Parser’ must verify and ensure that no syntax error occurred and also the existence of the objects while processing performed by the ‘Tracker’ and ‘Sequencer’ rules before transmitting all the obtained and necessary properties by ‘Linker’ rule in order to establish the relationship between the concerned objects.

4. **Linker**: This rule provides a relation between different elements where needed. It is applied only to establish the association between objects. However, this rule must determine which the transformation object must to link, i.e., the ‘Linker’ rule requires the inputs sent by the ‘Parser’ rule which provides the full reference to finally establish the relationship and its cardinalities based on the list values obtained by the ‘Sequencer’ rule between objects.

5. **Controller**: This rule verifies and controls if the semantics of the state machine can be supported.

6. **Atomicity**: This rule preserves the action atomicity of models. It must ensure and verify the consistency and reliability of the obtained results during the execution of each model.
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 transformation rules are used to establish the various models of patterns, as well as the transformation of one model into another either from PIM to PIM and/or PSM to PSM.

This example presents a general overview of the PIM to PIM models interactive system by applying model transformation rules between Domain model and Task model of POMA architecture [1] for a laptop platform.

The Figure 2 shows The [POMA.PIM]-independent Domain model which is obtained by composing patterns and applying the composition rules [12] and [15].

![Figure 2: UML class diagram of a PIM Domain model](image)

The Figure 3 shows The [POMA.PIM]-independent Task model which is obtained by composing patterns and applying the composition rules [12] and [15].

![Figure 3: UML class diagram of a PIM Task model](image)

Table 1 shows the transformation rules for the Domain model and Task model for laptop platforms to the unified PIM model.

<table>
<thead>
<tr>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
<th>Transformation Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>‘Login’</td>
<td>Class</td>
<td>Navigation</td>
<td>Tracker, Controller, Atomicity</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>‘Login’</td>
<td>Class</td>
<td>Navigation</td>
<td>Sequerencer, Controller, Atomicity</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>‘Login’</td>
<td>Class</td>
<td>Navigation</td>
<td>Parser, Controller, Atomicity</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>‘Login’</td>
<td>Class</td>
<td>Navigation</td>
<td>Linker, Controller, Atomicity</td>
<td></td>
</tr>
</tbody>
</table>
After the applied transformation rules, the PIM Domain model and PIM Task model are obtained together in one unified UML class diagram PIM model for a given platform – Figure 4.

The same design that has done for other PIM and PSM models such as: Dialog model, Presentation model and Layout model of POMA architecture.

5 Conclusion and further work

In this paper, a novel practical transformation rules for multi-platforms POMA architecture are introduced for interactive systems engineering. Then, POMA proposed six transformation rules that can be applying to inter-relate PIM models together and PSM models together. These rules allow maintaining the tracking structures of all class instances and for association populations, enforcing the event ordering, preserving the atomicity of the actions, supporting the state machine semantics.

Among the next steps required to develop POMA are:

- Description of a process for the generation of a source code from the five PSM models of POMA;
- Development of a tool 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.

6 References


[10] C. Alexander, the Timeless Way of Building, Oxford University Press, 1979, New York: (NY), USA.


