Transition from precisely defined requirements into draft architecture as an MDA realisation

Jacek Bojarski, Tomasz Straszak,
Albert Ambroziewicz, and Wiktor Nowakowski

Warsaw University of Technology,
Warsaw, Poland
{bojarsj1,straszat,ambrozia,nowakoww}@iem.pw.edu.pl
http://www.redseeds.eu/

Abstract. Automation and reuse are two elements which introduced into the software development process can significantly improve chances for success. Automation can be achieved by defining and using certain automatic transformations between artefacts (models and code) produced during the software lifecycle. These transformations encapsulate certain important knowledge that can be reused in future projects. In this paper we describe a method for combining automatic MDA style transformations with their reuse. An important factor is that this combination is introduced even from the highest level of abstraction which is expressed through requirements. Precisely (yet understandably for "ordinary people") defined requirements can be transformed automatically into draft architectural models. The rules of this transformation contain certain decisions on the target architectural framework. This paper presents ways to reuse these decisions on the basis of only the current requirements model.

1 Introduction

On every step of the software development process, numerous artefacts are produced and used. According to the Model Driven Architecture (MDA) concept (see [7]), all of these artefacts should be integrated into the software lifecycle by automatic transformations that are used to convert one model into another.

Within the ReDSeeDS[1] project, efforts are underway to integrate the Requirements Specification, Architectural Model, Detailed Design Model and Code through traceability links in order to make it easy for further change management and reuse. To create traces automatically, well defined and unambiguous transformations between models should be performed. Such transformation should operate on source and target products expressed with precisely defined languages. Transformations encapsulating certain important knowledge on the target model, gathered in transformation profile and defined for concrete technology, style and methodology may be applied during construction of another software product.

[1] Requirements Driven Software Developement System, www.redseeds.eu

As the problem of architecture (PIM equivalent) to detailed design (PSM equivalent) and detailed design to code synchronisation seems to be well investigated, we focus on the requirements (CIM equivalent) to architecture (PIM equivalent) transition. We need an exact definition of transformation from CIM to PIM, as well as eligible languages for expressing requirements and architecture, which will be performed in realisation of the MDA.

Having a software product constructed in the way described above, reuse of its part on the basis of a particular requirements element becomes possible. Another advantage of such solution is generalisation of architectural knowledge as it is contained in the transformation profiles. An architectural style implemented in a transformation profile generalises architectural experience captured during work within earlier projects. Applying a transformation profile is in fact equivalent to reusing knowledge from past projects.

This paper focuses on transitions from precisely defined requirements to draft architectural models based on a semiformal requirements specification language. Section 2 describes the idea of CIM to PIM transformations based on requirements. Section 3 includes a brief description of languages and tools used in realisation of this idea. The next section introduces the method of performing transformations. Section 5 describes an example realisation of a CIM to PIM transformation with appropriate validation results. In section 6, conclusions and future work are presented.

2 Performing transformations within the MDA

According to the MDA Guide ([7]) CIM is a view of a system from the computation independent viewpoint. The role of CIM is *"bridging the gap between those that are experts about the domain and its requirements on the one hand, and those that are experts of the design and construction of the artifacts that together satisfy the domain requirements, on the other"*. The computation independent model, sometimes called the domain model or the business model, should provide a common vocabulary to be used in other models (PIM and PSM). It shows the system in the environment in which it will operate and specifies requirements for the system. Another important statement about CIM says that *"CIM requirements should be traceable to the PIM and PSM constructs that implement them, and vice versa"*. Such traces to the lower level models enable analysing dependency of software architecture and design on requirements.

Fulfilling such requirements for a language describing CIM, together with formalising the structure of the domain vocabulary and the requirements, enables to define transformations from precisely defined requirements to draft architectural models. To be able to perform such transformations, we should also specify the target language, ie. the language for describing architectures. Such language should be suitable for expressing architectural issues and precisely defined for transformation purposes. The transformation itself, can then handle the issue of creating traces from individual requirements to these models or artefacts that realise (fulfill) them.

The idea of such transformation in the context of MDA is shown in Figure 1. In the proposed solution, CIM which is the source for transformation to architecture is modelled in a specific Requirements Specification Language (RSL). It is designed specifically for this purpose and it can be argued, that it fulfills requirements defined in the MDA Guide. PIM and PSM models are expressed in a precisely defined subset of UML (Software Development Specification Language - SDSL). Rules for transformation from CIM to PIM and from PIM to PSM are defined in another special-purpose language – MOLA.

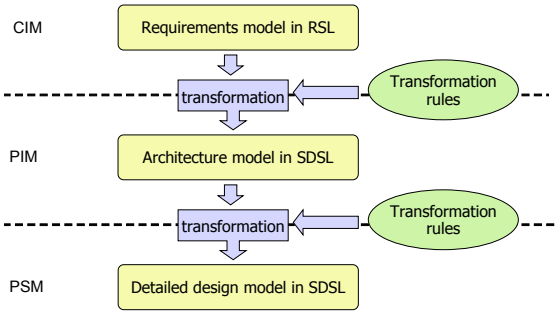

Fig. 1. ReDSeeDS transformations in context of MDA

3 Languages supporting the proposed solution

To be able to express all artefacts in a software project in the way described above, there is a need of introducing new or adapting the existing languages suitable for transformation purposes. The three languages mentioned in the previous section (RSL, SDSL, MOLA) compose into the Software Case Language (SCL) which is sufficient to express all aspects needed in a software project. The following sections describe briefly the main features of RSL and SDSL. Detailed description of the MOLA language developed at the University of Latvia can be found in [6]. Full specification of the language can be found in [3, 2].

3.1 Requirements Specification Language

Requirements Specification Language (RSL) is the MOF[2]–based language designed for precise and unambiguous definition of requirements. RSL is specified in a meta-modelling language MOF ([8]). Some elements of RSL are adapted from UML ([9]). For example, RSL uses an extended UML Use Case model, and UML Activity and Interaction models are the basis for the graphical representations of scenarios. For more details please refer to the language specification in [4].

[2] OMG's Meta Object Facility

The main advantages of RSL are clear separation of "requirements as such" from their representations and separating sequence of actions performed between the user and the system from description of the environment (behavioural and statical requirements). Environment description is stored in the vocabulary model, which holds all notions used in the behavioural requirements together with their definitions. Additionally, RSL provides different representations of the same information suitable for different audience. For example, textual representation of scenarios seems to be most suitable for stakeholders. Activity representations, focused on flow of control within the use case, can be preferred by analysts. Interaction representation shows precise dialogue between the user and the system needed by designers and developers. Figure 2 shows simple example of a requirements model, a vocabulary model and a graphical representations of scenarios (Activity and Interaction representations). More details can be found in [10].

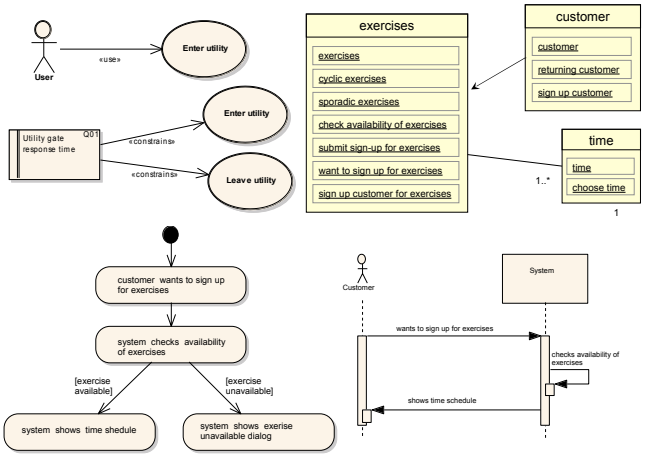

Fig. 2. RSL example

RSL is a semi-formal language. Requirements can be expressed in an informal way, using natural language or in more formal way through a constrained language. RSL's constrained language is based on simple SVO(O)[3] grammar[5]. Precisely defined requirements, expressed by constrained language scenarios, together with an external vocabulary enable the possibility of processing and automatic transformation of requirements into the Platform Independent Model (draft architecture).

3.2 Software Development Specification Language

Software Development Specification Language (SDSL) is a subset of UML containing just selected elements. It aims to express architectural and detailed design

[3] Subject – Verb – Direct Object(–Indirect Object)

models generated through ReDSeeDS transformations basing in RSL compliant requirements (see Figure 3). Such models can be manually extended with other UML models (not included in SDSL).

SDSL is defined in [3] as a UML profile and is composed of UML facilities for architectural and detailed design modelling. Both profiles are divided into a behavioural and a static part. These two parts contain UML elements that form subset of UML sufficient for being a target of a transformation from requirements. The behavioural models translate from functional requirements (use cases) and the static models translate from the vocabulary.

4 Method for transforming precisely defined requirements into architectural models

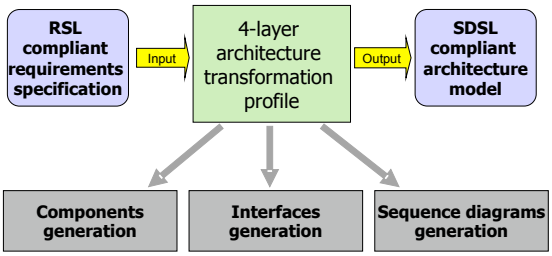

Fig. 3. Overview of requirements to architecture transformation

To validate the process of performing such transformations, an example transformation profile for a chosen architectural style was implemented (see section 5 for details). Figure 3 shows overview of this process, described below.

The input model for the transformation is a complete RSL-compliant requirements specification. Such a specification should contain requirements units (use cases) logically structured in requirements packages. Every use case should have its representation in the form of constrained language scenarios. Such scenarios are sequences of SVO sentences, used for expressing user – system interaction, and control sentences, depicting flow of control within a use case (between alternate scenarios) and between use cases. Every occurrence of a notion or a phrase in an SVO sentence should be connected to its definition in the vocabulary. All vocabulary entries should be logically structured in vocabulary packages.

When a complete requirements specification is available, a transformation profile can be applied to it. During the transformation process, certain SDSL elements of the output architectural models are generated. These elements (components, interfaces, classes, packages and sequence diagrams together with messages) are generated on the basis of requirements specification elements (requirements and vocabulary packages, requirements, SVO scenarios and vocabulary notions with phrases) according to the transformation algorithm defined in the transformation profile.

The output model of the transformation is an SDSL compliant logical architecture, relevant for the structure and behaviour of the source model. This model contains a component model reflecting the logical structure of requirements and domain vocabulary together with DTOs[4] exchanged between components. The architectural model contains also sequence diagrams for this model corresponding to user-system interaction defined in requirements scenarios. All messages exchanged between components in sequence diagrams are generated as operations in the corresponding interfaces thus keeping the target model coherent.

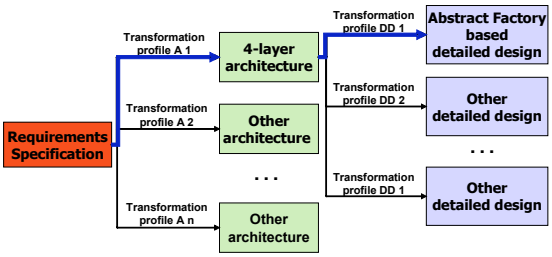

Fig. 4. Family of transformation profiles

The described process does not take into account the selection of the transformation profile. In a real life project there is a need of selecting a suitable architectural style, that reflects also the non-functional requirements. This decision can be guided through a reuse process, where an appropriate transformation profile is selected from the whole family of profiles. Their relevance is determined by comparing the current requirements specification with the past software specifications that were sources to these legacy transformation profiles. Figure 4 gives some idea on the diversity of possible profiles originating from a single requirements specification. The decision of choosing from the found transformation profiles should be done manually by the architect on the basis of non-functional requirements and the architect's experience. Similarly, the designer should be able to select a transformation from PIM to PSM for the chosen technologies, possibly also the most suitable for fulfilling the non-functional requirements.

5 Examplary transformation profile for the 4-tiers architectural style

A commonly used style for building contemporary business systems is based on tiers. We have chosen a 4-tiers framework for the target of our CIM to PIM transformation. The four tiers include Presentation, Application Logic, Business Logic and Data Access/Storage. Below we briefly describe the rules for transforming requirements expressed in RSL into an SDSL model conforming to this 4-tiers style. These rules are presented in detail in [2].

[4] Data Transfer Objects

The Presentation tier, contains only one component with one interface and plays the role of a proxy between the user and the system. The Application Logic tier contains components, which reflect use cases packages in the requirements specification. Interfaces provided by each of these components come from use cases included in the specific use cases package. Operations of these interfaces are generated on the basis of predicate phrases in SVO scenario sentences directed from the actor to the system. This tier is thus responsible for realisation of functional requirements. The Business Logic tier contains components, which reflect the packages in the vocabulary. Packaged vocabulary notions are the basis for interfaces provided by a specific business tier component. Operations of these interfaces are generated on the basis of predicate phrases in SVO scenario sentences directed from the system to itself. This tier is thus responsible for realisation of business rules defined within the vocabulary. Data Access tier is responsible for manipulating on the data source. It is built with components, reflecting notions packages the same as for the Business Logic tier. These components have simple DAO[5] interfaces for each notion in the specific notions package. For all notions used in the SVO scenario sentences, DTO classes with associations reflecting notion links are generated. Having this architectural structure generated, its behaviour can be generated on the base of control flow expressed in use case scenarios. Sequence diagrams for architecture dynamics reflect all scenarios in the requirements specification. Every message in a sequence diagram is transformed from one or more SVO sentences. Full description of these rules and the their MOLA implementation can be found again in [2].

6 Conclusions and future work

This paper introduces the idea of reusable transformations that encapsulate knowledge about transformation from requirements to design. A set of precisely defined languages described in section 3 allows for construction of coherently mapped software artifacts on different levels of abstraction. Transition from an abstract description of the problem (requirements and domain vocabulary defined as CIM) to that problem's solution (draft architecture model defined as PIM) can be performed automatically using standardised transformation profiles. Well formed profiles can be treated as kind of reusable artifacts generalising architectural knowledge and experience.

The validation of the proposed idea proves that the MDA concept of CIM to PIM transitions for the exemplary profile described above is possible. Prototype tools used in experiments, allowed for generating sensible architectural models, usable by developers. Industry people found this automation useful and affirmed to use it in the future. Current efforts concentrate on developing a comprehensive Engine allowing for convenient formulation and execution of models and their transformations. This Engine is also being prepared to have certain capabilities of storing and querying for the stored transformation profiles.

[5] Data Access Object

Experience gained so far puts also certain emphasis on non-functional requirements (NFR) in the process of generating architecture. The RSL has still a weak support for NFRs. Therefore capturing relations between NFRs and transformation rules for appliance them in the architecture is not possible. Thus, at present, the generated architecture, which fully reflects all the functional and vocabulary requirements, has to be completed with NFRs manually. Future research is necessary to introduce the ability of reflecting also the non-functional ones. One of the solutions to be investigated includes adapting the NFR Framework with softgoals as described in [1]), and extended onto use cases in [11]. The final step would be to include such well formed non-functional requirements into the transformation process to make such transformations parameterisable.

Acknowledgments. This work is partially funded by the EU: Requirements-Driven Software Development System (ReDSeeDS) (contract no. IST-2006-33596 under 6FP). The project is coordinated by Infovide, Poland with technical lead of Warsaw University of Technology and with University of Koblenz-Landau, Vienna University of Technology, Fraunhofer IESE, University of Latvia, HITeC e.V. c/o University of Hamburg, Heriot-Watt University, PRO DV, Cybersoft and Algoritmu Sistemos.

References

1. Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*, volume 5 of *International Series in Software Engineering*. Springer, October 1999.
2. Audris Kalnins et al. Reusable case transformation rule specification. Project Deliverable D3.3, ReDSeeDS Project, 2007. www.redseeds.eu.
3. Audris Kalnins et al. Reuse-oriented modelling and transformation language definition. Project Deliverable D3.2.1, ReDSeeDS Project, 2007. www.redseeds.eu.
4. Hermann Kaindl et al. Requirements specification language definition. Project Deliverable D2.4.1, ReDSeeDS Project, 2007. www.redseeds.eu.
5. Ian M Graham. Task scripts, use cases and scenarios in object-oriented analysis. *Object-Oriented Systems*, 3(3):123–142, 1996.
6. Audris Kalnins, Janis Barzdins, and Edgars Celms. Basics of model transformation language MOLA. In *Workshop on Model Driven Development (WMDD 2004)*, 2004.
7. Joaquin Miller and Jishnu Mukerji, editors. *MDA Guide Version 1.0.1, omg/03-06-01.* Object Management Group, 2003.
8. Object Management Group. *Meta Object Facility (MOF) 2.0 Core Specification, Final Adopted Specification, ptc/03-10-04*, 2003.
9. Object Management Group. *Unified Modeling Language: Superstructure, version 2.0, formal/05-07-04*, 2005.
10. Michał Śmiałek, Jacek Bojarski, Wiktor Nowakowski, Albert Ambroziewicz, and Tomasz Straszak. Complementary use case scenario representations based on domain vocabularies. *Lecture Notes in Computer Science*, 4735:544–558, 2007.
11. G. Sousa, S. Soares, P. Borba, and J. Castro. Separation of crosscutting concerns from requirements to design: Adapting the use case driven approach. In *Early Aspects Workshop at AOSD*, 2004.