Luís Ferreira Pires and
Slimane Hammoudi (Eds.)

Model-Driven Enterprise Information Systems

Proceedings of the
2nd International Workshop on
Model-Driven Enterprise
Information Systems,
MDEIS 2006
In conjunction with ICEIS 2006
Paphos, Cyprus, May 2006

INSTICC PRESS
Portugal

Volume Editors

Luís Ferreira Pires
Faculty of Electrical Engineering, Mathematics & Computer Science
University of Twente
Enschede, The Netherlands
l.ferreirapires@ewi.utwente.nl

and

Slimane Hammoudi
ESEO
Grande Ecole d'Ingénieurs Généralistes
en électronique, informatique, télécoms et réseaux
Angers, France
slimane.hammoudi@eseo.fr

2nd International Workshop on
Model-Driven Enterprise
Information Systems - (MDEIS 2006)
Paphos, Cyprus, May 2006.
Luís Ferreira Pires and
Slimane Hammoudi (Eds.)

Printed in Portugal

ISBN: 972-8865-56-2
ISBN (13 digits): 978-972-8865-56-6
Depósito Legal: 241852/06

Foreword

This volume contains the proceedings of the Second International Workshop on Model-Driven Enterprise Information Systems (MDEIS) held in conjunction with the 8th International Conference on Enterprise Information Systems (ICEIS) in Paphos, Cyprus. The main aim of this workshop is to serve as a forum for researchers and practitioners to meet and to share expertise in Model-Driven Architecture (MDA) and its application to Enterprise Information Systems.

The potential benefits of MDA are reduction on development costs, improvement of software quality, reduction of maintenance costs and the support for controlled evolution of IT systems. MDA has been applied in many application areas, such as real-time and embedded systems, and telecommunication systems, and, more recently, to the development and integration of enterprise information systems. The goal of this workshop is to bring together people working on MDA techniques and tools, and applying them on enterprise information systems, so that they can exchange their experience with the use of MDA, create new ideas, evaluate and improve MDA and spread its use.

We have received 19 paper submissions, and 9 papers have been accepted for publication and oral presentations. All selected papers are of high quality, thanks to the professionalism of all authors, reviewers and program committee members.

The selected papers are very good illustrations of the three main topics in the Model-Driven Architecture that are currently under intense research:

- Modeling and Metamodeling;
- Transformations;
- MDA Applications.

The papers in this volume have been grouped around these topics (three papers for each topic).

We would like to take this opportunity to thank the people who have contributed to MDEIS 2006. We wish to thank all authors and reviewers for their valuable contributions to MDEIS 2006, and we wish them a successful continuation of their research. Finally, special thanks to

Joaquim Filipe and Vitor Pedrosa for their hard work in making the workshops and this volume possible.

We wish all authors and attendees an exciting workshop, and a pleasant stay in the beautiful place of Paphos.

May 2006

Luís Ferreira Pires
Slimane Hammoudi

Workshop Chairs

Luís Ferreira Pires
Faculty of Electrical Engineering, Mathematics & Computer Science
University of Twente
Enschede, The Netherlands
l.ferreirapires@ewi.utwente.nl

and

Slimane Hammoudi
ESEO
Grande Ecole d'Ingénieurs Généralistes
en électronique, informatique, télécoms et réseaux
Angers, France
slimane.hammoudi@eseo.fr

Program Committee

João Paulo A. Almeida, Telematica Instituut, The Netherlands
Jean-Michel Bruel, University of Pau, France
Jerome Delatour, ESEO, France
Anastasius Gavras, Eurescom, Germany
Jeffrey G. Gray, University of Alabama at Birmingham, USA
Nicolas Guelfi, University du Luxembourg, Luxembourg
Sune Jakobsson, Telenor, Norway
Kai Koskimies, Tampere University of Technology, Finland
Santosh Kumaran, IBM, USA
Zhiming Liu, United Nation University, Macau
Denivaldo Lopes, Federal University of Maranhão, Brazil
Ilia Petrov, University of Erlangen-Nuernberg, Germany
Pascal Roques, Valtech Training, France
Marten van Sinderen, University of Twente, The Netherlands
Richard Mark Soley, OMG, USA
Jean Louis Sourrouille, INSA , University de Lyon, France
Andreas Tolk, Virginia Modeling, Analysis and Simulation Center, USA
Antonio Vallecillo, University de Málaga, Spain
François Vernadat, European Commission, Luxembourg

Table of Contents

Papers

Modeling

Transformations

Applications

Papers

Modeling

Towards Rigorous Metamodeling

Benoît Combemale[1], Sylvain Rougemaille[1], Xavier Crégut[1], Frédéric Migeon[1], Marc Pantel[1], Christine Maurel[1] and Bernard Coulette[2]

[1] FéRIA-IRIT-LYRE
118, route de Narbonne
F-31062 Toulouse Cedex 9
{sylvain.rougemaille, frederic.migeon, christine.maurel}@irit.fr
{benoit.combemale, xavier.cregut, marc.pantel}@enseeiht.fr

[2] GRIMM ISYCOM
5, allée Antonio Machado
F-31058 Toulouse Cedex 9
bernard.coulette@univ-tlse2.fr

Abstract. MDE has provided several significant improvements in the development of complex systems by focusing on more abstract issues than programming. However, improvments are needed on the semantic side in order to reach high-level certification such as the one currently required for critical embedded systems (which will also probably be required in the near future for Information Systems as application of Basel II kind of agreements). This paper presents different means to specify models semantics at the metamodel level. We will focus on the definition of executable SPEM-based development process models (workflow related models) using an approach defined for the TOPCASED project.

1 Introduction

Model-Driven Engineering (MDE) has succeeded in establishing a new, more abstract, approach to large scale system development. A system can be described using many different models which are related to each other using model transformations. The key point is to use as many different models as life-time or technology aspects in the system. The main difference with programs is that model focus on the abstract syntax whereas programs focus on the concrete syntax. A single model can then be represented using different graphical or textual concrete syntaxes. For data-centred systems, the level of abstraction thus provided led to significant improvements and seems to correspond to an adequate semantic level. For computation-centred systems, further steps are required in order to give a precise enough account of the dynamic aspects of the models.

This contribution gives some insights on approaches for defining metalevel model semantics derived from the work done by the programming language community. The evocated experiments take place in the TOPCASED project [1] whose purpose is to define and implement a MDE-centred CASE tool for critical embedded software and hardware systems. The certification authorities for TOPCASED application domain (aeronautic, space, automotive...) require high quality system validation approaches which are currently based on formal tools. Currently, Information Systems do not require this

kind of certification. However, we can guess that, in the near future, the software developed for Basel II level IS will also follow this level of requirements.

The TOPCASED toolkit aims at easing the definition of new DSL or modeling languages by providing metalevel technologies such as concrete syntax (both textual and graphical) editor generators, static validation and dynamic execution of models. This contribution will describe how semantic considerations designed for programming languages can be integrated in the MDE approach. We will focus on one of the available technologies for creating executable models; the other ones will be reported in forthcoming publications.

As an example, we apply our proposal for the modeling of a very simplified process description language (PDL). This PDL provides the concept of processes (*Process*) composed of activity (*Activity*) sequences representing the various tasks which must be realized during the development. These activities may be connected using a relation of precedence (*Precedes*) which makes it possible to indicate a partial order *start-to-start*, *finish-to-start* and *finish-to-finish* (*PrecedenceKind*). This kind of example is very close to the workflow-based modeling of IS.

2 Syntax in Metamodeling

2.1 Abstract Syntax Definition

The abstract syntax of a modeling language is the structural expression of its concepts and the relations which bind them. Metamodeling languages such as the OMG standard MOF (Meta Object Facility) [2], provide sets of elementary entities and relations in which terms we can describe our own metamodel. Nowadays, the definition of this syntax is well mastered and supported by many metamodeling environments (Eclipse/Ecore [3], GME/MetaGME [4], AMMA/KM3 [5] and XMF-Mosaic/Xcore [6]).

To describe the abstract syntax of our SimplePDL, we use the Ecore editor from the TOPCASED project. It is a graphical editor that allows the description of the abstract syntax. For SimplePDL, we draw the metamodel of figure 1. *Process*, *Activity* and *Precedes* are instances of the EClass metaclass of Ecore. Their characteristics, such as, e.g., *name*, are described as EAttribute and their relationships are defined as EReference. This metamodel will be used as a basis for the various experiments. First of all, we would like to have a concrete syntax to be able to define models conforming to SimplePDL.

2.2 Concrete Syntax Design

Concrete syntax provides a formalism, graphical or textual, to handle concepts of the abstract syntax and thus to describe “instances” of the abstract syntax. The definition of ad hoc concrete syntaxes is well mastered, indeed many projects exist for this purpose which are mainly built upon EMF (Eclipse Modeling Framework) : GMF[3], Merlin

[3] Generic Modeling Framework, `http://www.eclipse.org/gmf/tutorial/`

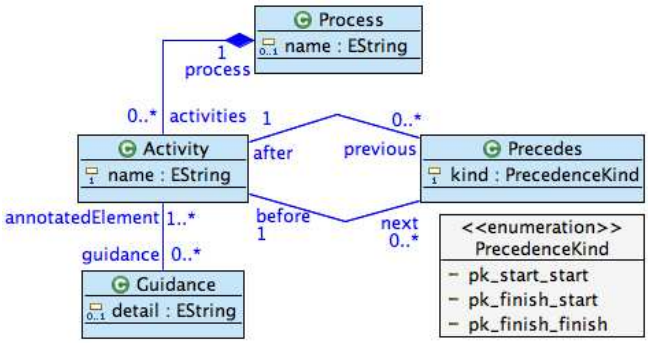


Fig. 1. The Ecore metamodel of our Simple PDL.

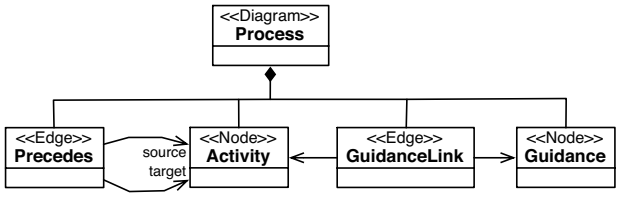


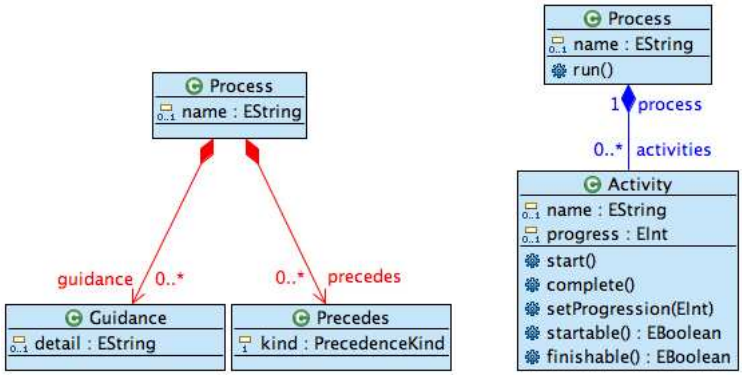
Fig. 2. Configurator model.

Generator[4], GEMS[5], TIGER [7], etc. The current challenge aims at being able to generate automatically a concrete syntax from an already defined abstract syntax [4, 6]. This generative approach, in addition to its generic qualities, would allow to standardize the construction of concrete syntaxes.

The TOPCASED environment offers a tool called "graphical editor generator" that allows to define a graphical concrete syntax and the associated editor for an Ecore model. The generation process takes place after the generation of the textual syntax (XML) provided by EMF [3]. It is based on the definition of the items of the graphical formalism (concrete syntax) and its mapping to the base Ecore model (abstract syntax). These two things are described in the *configuration model* (*configurator*) that offers strong possibilities for the personalization of this concrete syntax.

For SimplePDL, we first defined the model of our concrete syntax (figure 2). *Activity* and *Guidance* are defined as *Node* (boxes). *Precedes* is defined as an *Edge*, a connection between two nodes. *Process* is represented as a *Diagram*, which is a package that will contain the other items. The concrete syntax may need additional items that do not correspond to any abstract concept. For example, we need to add *GuidanceLink* as an *Edge* to connect a *Guidance* to the descrided *Activity*. *GuidanceLink* do not correspond to any concept of SimplePDL but is required to link a guidance to an activity (EReference named *guidance* on the base metamodel, figure 1). Please note that concepts of abstract syntax (figure 1) and concepts of concrete syntax (figure 2) are different con-

[4] `http://sourceforge.net/projects/merlingenerator/`
[5] Generic Eclipse Modeling System, `http://sourceforge.net/projects/gems/`



(a) Concrete syntax (b) Operational semantics

Fig. 3. Extensions of the SimplePDL abstract syntax.

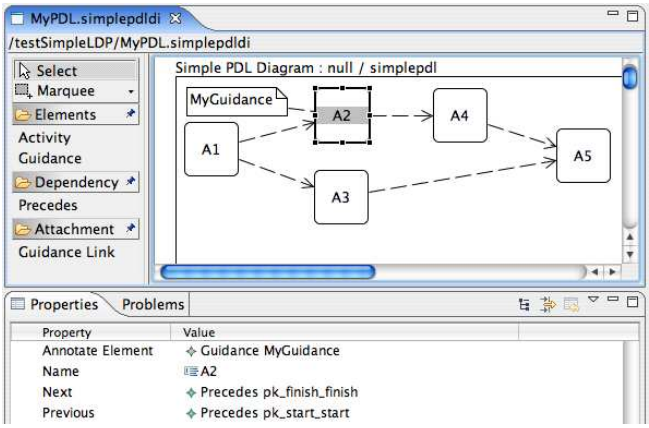


Fig. 4. Generated editor to provide a graphical concrete syntax to SimplePDL.

cepts that have to be mapped to each other. We used the same name when the mapping was obvious.

To be able to use the TOPCASED editor generator, we had to extend our abstract syntax (fig. 1) to add two containment references, one between *Process* and *Guidance*, and the other between *Process* and *Precedes* (fig. 3a). This is required to be able to put the corresponding graphical items (*Activity* and *Guidance*) in the *Process* package.

Figure 4 shows the generated editor. All the concepts of the configurator model are available using the palette. Clicking on a palette item and adding it on the diagram, creates a graphical feature (node, edge) and instantiates the corresponding SimplePDL metaclass according to the configurator model.

3 Semantics in Metamodeling

In the scope of MDE there are currently a lot of languages and techniques to define the abstract and concrete syntax of a modeling language. However, these techniques do not take into account the description of the precise meaning of the concepts provided by a modeling language. Consequently, the semantics of these languages has to be defined by the use of additional techniques allowing to enrich their abstract syntax. These problems have previously been handled by the programming language community. It is not surprising that the same approaches can also be used here. We can separate modeling language semantics into three categories (as defined for programming languages): axiomatic, operational and denotational (left for further work). Each of them can be applied at different levels.

3.1 Axiomatic Semantics

The axiomatic semantics is based on mathematic logic and allows to define correctness for the use of programming language constructions. The principle is to define axioms and deduction rules to express the meaning of such constructions according to invariants, pre and post-conditions. Thus it gives the precise semantics of every program written in this language along with a correctness proof scheme. In a model-driven approach, we restrict this semantics to models static analysis, which allow us to check the correctness of models structure. This vision of axiomatic semantics can be added by means of Well-Formed-Rules (WFR), which are expressed on the metamodel and have to be respected by the models. The OMG recommends the use of OCL (Object Constraint Language) [8] for the expression of WFR on metamodels. The metamodel WFR can be seen as a mean to reduce the number of valid models. Thus, one can use OCL checkers (e.g., Use [9]) to verify the correctness of a model in accordance with each of the WFR expressed on its metamodel.

One can also check whether a model satisfy its WFR or not by means of a declarative transformation language such as ATL (Atlas Transformation Language) [10]. The idea is to define transformation rules that match errors and generate a diagnostic model containing much more details than the Boolean return value of an OCL checker. The details concerning the errors depend on the diagnostic metamodel. This technique has been proposed by the ATLAS team and carried out thanks to ATL [11]. ATL transformation rules are defined to detect the negation of a WFR and generate the corresponding diagnostic model (fig. 5).

In the scope of our language (SimplePDL), several constraints have to be defined to guarantee the consistency of the models which conform to the metamodel. As an example we proposed the following rules :

“ An activity must not precede itself ” :

```
context Precedes inv :
       self.before <> self.after
```

“ A process must not contain two activities with the same name ” :

```
context Process inv :
       self.activities->forAll(a1, a2 : Activity |
                        a1 <> a2 implies a1.name <> a2.name)
```

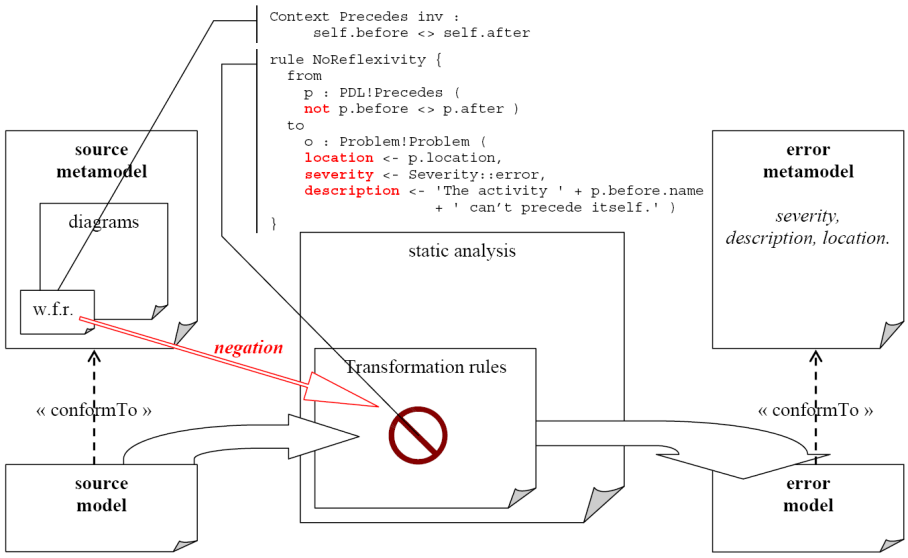


Fig. 5. Checking model with ATL.

3.2 Operational Semantics

The operational semantics allows to precisely describe the dynamic behavior of the constructions of a language. In MDE, it aims to express the behavioral semantics of a metamodel and thus build executable conforming models. For this purpose, two approaches are available. First of all, the one which is closer to the operational semantics in programming languages consists in the definition of transformations between two execution states of a model. The whole set of transformations, written in conformance to the metamodel, defines the behavior of models. The second one consists of describing the behavior of each concept of the metamodel in an imperative way using metaprogramming languages such as Kermeta [12], xOCL [6] or an action language such as AS-MOF [13].

Our first experimentation is related to Kermeta which is defined as an executable metamodeling language, or as an object oriented metaprogramming language, i.e., it allows to describe metamodels whose models are executable. Kermeta relies on the Ecore metamodeling language, it has been defined as a ”weaving” between a behavioral model and the Ecore metadata model [12]. The Kermeta metamodel is composed of two packages. The first one called *core* corresponds to Ecore. The second one called *behavior* is built as a metaclass hierarchy representing the expressions that constitute the body of the *operation* features defined in the *core* package. Thus, Kermeta allows to specify the structure of a metamodel as well as its behavior.

Kermeta is integrated as a plug-in to the Eclipse IDE, and it provides a generation tool *Ecore2Kermeta* which has allowed us to translate our SimplePDL metamodel (fig. 1) to a Kermeta version. This version of our metamodel has been used as a basis

for the programming of the SimplePDL models behavior. In order to code this behavior, we have had to define precisely what is the execution of a SimplePDL model.

A *Process* is composed of *Activities*, which goes through different states during the enactment of the *Process*: not started, in progress and completed. In order to represent those states we have added the *progress* attribute to the *Activity Eclass*. Thus, its progression rate value corresponds to its three possible states : -1: not started ; [0..99]: in progress and 100: complete. A *Process* have been executed when all the contained activities are completed. The behavior of our SimplePDL *process* consists of authorizing users to set the values of activities progression rate, according to *precedes* relation, until they are all finished. The handling of the progression rate and the *precedes* link for each *Activity* implies the extension of our metamodel (fig. 1) in order to add the necessary operations (fig. 3b). Thus, the execution of SimplePDL processes was implemented in Kermeta as a loop proposing to the user the following choices :

– *Stop the process execution:* Quit the loop.

– *Start an enactable activity:* One selects the activity which can be started. An activity can start if the *startable* operation returns *True*, i.e., if it is an initial one, or if its preceding activities and the *Precedes* link which bind them to it allows to.

```
operation startable() : Boolean is do
    var start_ok : kermeta::standard::Boolean
    var previousActivities : seq Activity [0..*]
    var prevPrecedes : seq Precedes [0..*]

    if progress==-1 then
        // Getting the activities which have to be started
        prevPrecedes := previous.select{p | p.kind ==
                                    PrecedenceKind.pk_start_start }
        previousActivities := prevPrecedes.collect{p | p.before}
        start_ok := previousActivities.forAll{a | a.progress >= 0}
        // Getting the activities which have to be finished
        prevPrecedes := previous.select{p | p.kind ==
                                   PrecedenceKind.pk_finish_start }
        previousActivities := prevPrecedes.collect{p | p.before}
        start_ok := start_ok and
                    (previousActivities.forAll{a | a.progress==100})
        result := start_ok or (previous.size() == 0)
    else
        result := false
    end
end
```

The user chooses the activity he wants to start, then its *progress* is set to 0.

```
operation start() : Void is do
    progress := 0
end
```

– *Make the progression rate of a started activity evolve:* One selects the activities whose progression rate can evolve. Then, the user chooses the one whose progression he wants to increase and gives the progression percentage that will be added to the current rate (operation *setProgression*).

– *Finish an activity:* One selects all the activities that can be stopped, i.e., those whose *finishable* operation return *True*. *finishable* evaluate whether an activity can be stopped or not according to the precedences rules to which it is subjected (relation *Precedes*).

```
operation finishable() : Boolean is do
    var finish_ok : kermeta::standard::Boolean
    var previousActivities : seq Activity [0..*]
    var prevPrecedes : seq Precedes [0..*]
    // Activities must be started
    if progress < 100 and progress >= 0 then
        // Testing previous activities
        prevPrecedes :=
            previous.select{p | p.kind == PrecedenceKind.pk_finish_finish }
        previousActivities := prevPrecedes.collect{p | p.before}
        finish_ok := previousActivities.forAll{a | a.progress==100}
        result := (finish_ok or previous.size()==0)
    else
        result := false
    end
end
```

Then the user selects the one he wants to be finished.

```
operation complete() : Void is do
    progress := 100
end
```

This loop and the choices proposals are implemented in the body of the *run()* operation of the *Process* metaclass. This execution model describes the behavior of all the models which conform to our Kermeta metamodel (SimplePDL); it represents the operational semantics of our Process Description Language.

4 Related Work

The definition of a rigorous semantics for modeling languages is currently a crucial issue in the ”Model-Driven” world. We can note two works that deal with this particularly important problem.

The ISIS laboratory from the Vanderbilt University has been involved in model engineering for many years. They promote the principles of MIC (Model-Integrated Computing), which places models as center piece for the integrated software development. They are developing the GME tool [4], which allows to describe DSL for multi-aspect and hierarchical models. In this scope they face the same problem concerning the definition of precise semantics. They recently proposed to ”anchor” the semantics of a particular DSL into a well-defined and formal semantics model [14]: the ASM (Abstract State Machine) [15] using their transformation modeling language GReAT (Graph Rewriting And Transformation language) [16].

Xactium[6] is a company founded in 2003 whose objective is to provide practical solutions for the development of large software system based on model-driven principles. They developed the XMF-Mosaic tool [6], which allows to define DSL, to simulate and validate models thanks to an extension of the OCL language called xOCL (eXecutable OCL). It provides means to transform models and to define mapping between them and other features for handling models.

These works are very close to the objectives of the TOPCASED environment, i.e., proposing an adaptive modeling environment based on a generative approach (as GME, XMF), offering means of simulation, validation of models by the definition of rigorous semantics.

[6] `http://www.xactium.com`

5 Conclusion and Future Work

This paper advocates the need for more semantic consideration in MDE. We then present several approaches for the integration of these points which are derived from previous work from the programming languages community. We focus on the definition of executable models for a very small subset of the SPEM development process modeling language. This work was based on the use of the Kermeta tool which weaves the model semantics with the metamodel. Further work will detail the other approaches in order to gather engineering knowledge around the semantic MDE. For instance, we are studying the possibility to define denotational semantics. In the programming languages scope, this semantics describes instructions as mathematical objects (i.e., function, integer, tuples, truth value etc.). The main idea of denotational semantics is to associate each phrase of the language with the appropriate mathematical object and thus, to map syntactic domain to a well-defined semantic domain. Mathematical objects are called the *denotation* of syntactic phrases, which are themselves said to *denote* objects. We can say that this denotation is a kind of translation to the mathematics world.

We are foreseeing a similar approach to provide a rigorous definition of DSL semantics. The idea is to target a well-known and well-defined formal language instead of mathematical objects. The challenge is to define transformation from DSL to another language owned by a different technological space and that has a rigorous semantics. This is often called translational semantics [6]. Those technological bridges allow to profit from simulation, checking and execution tools provided by the targeted technological spaces. We are considering to us ATL to define transformations from our DSL to semantics models such as Petri nets, timed automata or transition systems.

We are also expecting to use model transformations to describe rewriting rules over models. Thus, we will be able to express operational semantics in a closer way to former Structural Operational Semantics defined for programming languages by Plotkin [17]. The main profit of this method is that semantics of a language is expressed in its own terms, i.e., there is no need of additional concepts except those related to the transformation language.

This work puts forward the fact that many different metamodels need to be defined in order to manage the various aspects of a system. All these models are differents but related. These relations must be managed in order to reduce the amount of work required for the definition of their semantics.

References

1. Farail, P., Gaufillet, P., Canals, A., Camus, C.L., Sciamma, D., Michel, P., Crégut, X., Pantel, M.: the TOPCASED project: a toolkit in open source for critical aeronautic systems design. In: Embedded Real Time Software (ERTS), Toulouse (2006)
2. Object Management Group, Inc.: Meta Object Facility (MOF) 2.0 Core Specification. (2003)
3. Budinsky, F., Steinberg, D., Ellersick, R.: Eclipse Modeling Framework : A Developer's Guide. Addison-Wesley Professional (2003)
4. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., IV, C.T., Nordstrom, G., Sprinkle, J., Volgyesi, P.: The generic modeling environment. In: Workshop on Intelligent Signal Processing, Budapest, Hungary (2001)

5. ATLAS: KM3 : Kernel metametamodel. Technical report, LINA & INRIA, Nantes (2005)
6. Clark, T., Evans, A., Sammut, P., Willans, J.: Applied metamodelling - a foundation for language driven development. version 0.1 (2004)
7. Ehrig, K., Ermel, C., Hänsgen, S., Taentzer, G.: Towards graph transformation based generation of visual editors using eclipse. Electr. Notes Theor. Comput. Sci **127** (2005)
8. Object Management Group, Inc.: UML Object Constraint Language (OCL) 2.0 Specification. (2003) Final Adopted Specification.
9. Richters, M., Gogolla, M.: Validating UML models and OCL constraints. In Evans, A., Kent, S., Selic, B., eds.: UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference. Volume 1939 of LNCS., Springer Verlag (2000) 265–277
10. Jouault, F., Kurtev, I.: Transforming models with atl. In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica (2005)
11. Bézivin, J., Jouault, F.: Using atl for checking models. In: GraMoT. (2005)
12. Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving executability into object-oriented metalanguages. In: LNCS, Montego Bay, Jamaica, MODELS/UML'2005, Springer (2005)
13. Breton, E.: Contribution à la représentation de processus par des techniques de métamodélisation. PhD thesis, Université de Nantes (2002)
14. Chen, K., Sztipanovits, J., Abdelwalhed, S., Jackson, E.: Semantic anchoring with model transformations. In LNCS 3748, S.V., ed.: Model Driven Architecture - Foundations and Applications, First European Conference (ECMDA-FA). (2005) 115–129
15. Gurevich, Y.: The abstract state machine paradigm: What is in and what is out. In: Ershov Memorial Conference. (2001)
16. Agrawal, A., Karsai, G., Kalmar, Z., Neema, S., Shi, F., Vizhanyo, A.: The design of a language for model transformations. Technical report, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37235, USA. (2005)
17. Plotkin, G.: A structural approach to operational semantics. Technical Report DAIMI FN-19, Department of Computer Science, Aarhus University, Denmark (1981)

Modeling ODP Correspondences using QVT

José Raúl Romero[1], Nathalie Moreno[2]and Antonio Vallecillo[2]

[1] Universidad de Córdoba (Spain), jrromero@uco.es
[2] University of Málaga (Spain), {vergara,av}@lcc.uma.es

Abstract. Viewpoint modeling is currently seen as an effective technique for specifying complex software systems. However, having a set of independent viewpoints on a system is not enough. These viewpoints should be related, and these relationships made explicit in order to count with a set of complete and consistent specifications. RM-ODP defines five complementary viewpoints for the specification of open distributed systems, and establishes correspondences between viewpoint elements. ODP correspondences provide statements that relate the various different viewpoint specifications, expressing their semantic relationships. However, ODP does not provide an exhaustive set of correspondences between viewpoints, nor defines any language or notation to represent such correspondences. In this paper we informally explore the use of MOF QVT for representing ODP correspondences in the context of ISO/IEC 19793, i.e., when the ODP viewpoint specifications of a system are represented as UML models. We initially show that QVT can be expressive enough to represent them, and discuss some of the issues that we have found when modeling ODP correspondences with QVT relations.

1 Introduction

Viewpoint modeling is gaining recognition as an effective approach for dealing with the inherent complexity of the design of large distributed systems. It comprises two major elements: model-driven development (MDD) on the one hand, and viewpoints on the other. The first one uses models as the key elements to direct the course of understanding, design, construction, deployment, operation, maintenance and evolution of systems. Models allow to state features and properties of systems accurately, at the right level of abstraction, and without delving into the implementation details—or even without giving a solution of how these properties can be achieved [1]. Viewpoints divide the system design according to several areas of concerns, and have been adopted by the majority of current software architectural practices, as described in IEEE Std. 1471 [2].

The Reference Model of Open Distributed Processing (RM-ODP) framework [3] provides five generic and complementary viewpoints on the system and its environment: *enterprise*, *information*, *computational*, *engineering* and *technology* viewpoints. They allow different stakeholders to observe the system from different perspectives [4]. In addition, five viewpoint languages define the concepts and rules for specifying ODP systems from these viewpoints.

ODP viewpoint languages are abstract, in the sense that the RM-ODP defines their concepts and structuring rules, but independently from any notation or concrete syntax to represent them. This allows focusing on the modeling concepts themselves rather

than on notational issues, and also allows the use of different notations, depending on the particular needs and on the appropriateness of the specific notation, e.g., Z for the information viewpoint, or Lotos for the computational viewpoint. The RM-ODP architectural semantics [5] deals with the representation of ODP concepts in different languages. However, this notation-independence may also bring along some limitations, e.g., it may hinder the development of ODP tools. The need to count with precise notations for expressing ODP specifications, and to develop ODP tools, motivated ISO/IEC and ITU-T to launch a joint project in 2004 which aims to define the use of UML for ODP system specifications [6]. This new initiative (hereinafter called UML4ODP) is expected to allow the development of tools for writing and analyzing ODP specifications, and to make use of the latest MDD practices for designing and implementing ODP systems. UML4ODP defines a set of UML Profiles for represent each of the viewpoint languages. In this way, the ODP viewpoint specifications are expressed as a set of UML models of the system. This initiative introduces very interesting benefits: ODP modelers can use the UML notation for expressing their ODP specifications in a standard graphical way, while UML modelers can use the RM-ODP concepts and mechanisms to structure their UML system specifications.

So far, most of the ODP community efforts have focused on the definition of the five viewpoints and their corresponding viewpoint languages. However, having a set of independent viewpoints on a system is not enough. These viewpoints should be somehow related, and these relationships made explicit in order to provide a *complete* and *consistent* specification of the system. The questions are: how can it be assured that indeed *one* system is specified? And, how can it be assured that no views impose contradictory requirements? The first problem concerns the conceptual *integration* of viewpoints, while the second one concerns the *consistency* of the viewpoints.

RM-ODP tries to address these issues by establishing correspondences between viewpoint elements. ODP correspondences do not form part of any one of the five viewpoints, but provide statements that relate the various different viewpoint specifications—expressing their semantic relationships. Hence, a proper ODP system specification consists of a set of viewpoint specifications, together with a set of correspondences between them. ODP does not provide however an exhaustive set of correspondences between viewpoints (ODP is silent about many of them), nor defines any language or notation to represent correspondences. But without explicitly representing them we cannot reason about them, nor properly tackle the integration and consistency issues mentioned above.

In this paper we explore the use of MOF QVT [7] for representing ODP correspondences in the context of UML4ODP, i.e., when the ODP viewpoint specifications of a system are represented as UML models. We show that QVT seems to be expressive enough to represent them, and discuss some of the issues that we have found when modeling ODP correspondences with QVT.

The structure of this paper is as follows. First, Section 2 provides a brief introduction to ODP, and also discusses some previous proposals for representing ODP correspondences. Section 3 provides a short introduction to QVT. Then, Section 4 presents our initial proposal, describing how to represent ODP correspondences with QVT. Section 5 discusses some the issues that we have found during our work. Finally, Section 6 draws some conclusions and outlines some future research activities.

2 ODP

RM-ODP is a reference model that aims at integrating a wide range of present and future ODP standards for distributed systems, maintaining consistency among them. The reference model provides the coordination framework for ODP standards, and offers a conceptual framework and an architecture that integrates aspects related to the distribution, interoperation and portability of software systems—in such a way that hardware heterogeneity, operating systems, networks, programming languages, databases and management systems are transparent to the user. In this sense, RM-ODP manages complexity through a "separation of concerns", addressing specific problems from different points of view.

In ODP terms, a *viewpoint* (on a system) is an abstraction that yields a specification of the whole system related to a particular set of concerns. ODP defines five viewpoints, covering all the domains of architectural design. These five viewpoints are:

- the **enterprise** viewpoint (EV), which is concerned with the purpose, scope and policies governing the activities of the specified system within the organization of which it is a part;
- the **information** viewpoint (IV), which is concerned with the kinds of information handled by the system and the constraints on the use and interpretation of that information;
- the **computational** viewpoint (CV), which is concerned with the functional decomposition of the system into a set of objects that interact at well-defined interfaces;
- the **engineering** viewpoint (NV), which is concerned with the infrastructure required to support distribution;
- the **technology** viewpoint (TV), which is concerned with the choice of technology used to implement the system and to connect it with its environment.

These viewpoints are of course mutually related, but no temporal order of their development is implied. They are (at least in theory) separately specified, and sufficiently independent to simplify reasoning about the complete system specification.

2.1 ODP Correspondences

ODP clearly states that a set of viewpoint specifications of an ODP system written in different viewpoint languages should not make mutually contradictory statements i.e., they should be mutually consistent.

The key to consistency is the idea of correspondences between different viewpoint specifications, i.e., a statement that some terms or structures in one specification correspond to other terms and structures in a second specification.

The requirement for consistency between viewpoint specifications implies that what is specified in one viewpoint specification about an entity needs to be consistent with what is said about the same entity in any other viewpoint specification. This includes the consistency of that entity's properties, structure and behavior.

The specifications produced in different ODP viewpoints are each complete statements in their respective viewpoint languages, with their own locally significant names,

possibly with different granularity, and so cannot be related without additional information in the form of **correspondence statements** that make clear how elements of different viewpoints are related, and how constraints from different viewpoints apply to particular elements of a single system to determine its overall behavior.

Correspondence statements relate the various different viewpoint specifications, but do not form part of any one of the five viewpoints. They fall into two categories [8]:

- Some correspondences are required in all ODP specifications; these are called **required correspondences**. If the correspondence is not valid in all instances in which the concepts related occur, the specification is not a valid ODP specification.
- In other cases, there is a requirement that the specifier provides a list of items in two specifications that correspond, but the content of this list is the result of a design choice; these are called **required correspondence statements**.

RM-ODP only provides required correspondences between the computational and engineering viewpoints, and between the engineering and the technology viewpoints. For the rest of the viewpoints, RM-ODP only states that elements of every viewpoint should be consistent with the specification of the corresponding elements in the rest of the viewpoints, and with the restrictions that apply to them. For instance, the elements of the information viewpoint should conform to the policies of the enterprise viewpoint and, likewise, all enterprise policies should be consistent with the static, dynamic, and invariant schemata defined by the information specification.

For illustration purposes let us include here some examples of ODP correspondences, as described in Part 3 of RM-ODP [8], the Enterprise Language [9], and in UML4ODP [6].

EC-1 Where there is a correspondence between enterprise and computational elements, the specifier has to provide, for each enterprise object in the enterprise specification, that configuration of computational objects (if any) that realizes the required behavior, and for each interaction in the enterprise specification, a list of those computational interfaces and operations or streams (if any) that correspond to the enterprise interaction, together with a statement of whether this correspondence applies to all occurrences of the interaction, or is qualified by a predicate.

CN-1 Each computational object that is not a binding object corresponds to a set of one or more basic engineering objects (and any channels which connect them). All the basic engineering objects in the set correspond only to that computational object.

CN-3 Where transparencies that replicate objects are involved, each computational interface of the objects being replicated corresponds to a set of engineering interfaces, one for each of the basic engineering objects resulting from the replication. Each of these engineering interfaces corresponds only to the original computational interface.

NT-1 Each engineering object corresponds to a set of one or more technology objects. The implementable standards for each technology object is dependent on the choice of technology.

2.2 Expressing Correspondences

Different authors have dealt with the problem of defining and expressing correspondences between viewpoints, mainly when trying to address the issue of viewpoint consistency checking. Some of the proposals, e.g., [10, 1], highlight the need to explicitly define and establish these correspondences but do not represent them as independent entities. Rather, they form part of the logical framework they define for checking the consistency of viewpoint specifications.

Other authors explicitly represent the correspondences, specially when viewpoint specifications are expressed as UML models, using different alternatives. One interesting possibility is the use of OCL to define relationships between the metamodel elements that represent the appropriate modeling concepts, as suggested by, e.g., [11]. This approach works very well when the correspondences are defined between all the instances of certain modeling concepts, e.g., when every computational interface corresponds exactly to one engineering interface (correspondence CN-2). However, there are cases in which correspondences need to be established between particular objects of an specification. The problem is that it is not possible at the metalevel to determine which particular objects should be related. Therefore, it is important that correspondences can be established between specific model elements, too.

UML 2.0 *abstraction dependencies*, possibly constrained by OCL statements, are the natural mechanism provided by UML to represent a relationship that relates two elements or sets of elements that represent the same concept at different levels of abstraction or from different viewpoints. Thus, ODP correspondences between viewpoint specifications (for example, between enterprise objects and information objects, or between enterprise policies and information schemata) can be expressed as UML abstraction dependencies between the corresponding UML model elements.

However, as suggested by [12, 13], viewpoint correspondences can also be used for other purposes, e.g., change management in multi-view systems. Change management implies consistent evolution of system specifications: if a view is modified for any reason (e.g., change of some business rules or some QoS requirements), several changes may need to be performed in other views in order to maintain the overall viewpoint consistency. In this context, correspondences act as “binds” that link together the related elements, transforming them if a change in one of them occurs, i.e., propagating the changes to maintain consistency.

UML abstraction dependencies show to be insufficient for these purposes. The main reasons are that they cannot store all the required information about the correspondence they represent, and because they can be used to express existence of the correspondence but not to enforce it. Therefore, Yahiaoui et al. define a new viewpoint, the *link* viewpoint, whose elements are “links” that establish binds between elements in different viewpoints. These links explicitly represent the ODP correspondences, and store the relevant information about the relationships between the views and the information related to each one (as attributes of the class that represents the link), thus guaranteeing traceability. A (change manager) tool has been developed for defining and enforcing these links, thus providing automated support for change management and propagation.

We do not think that such correspondences constitute another ODP viewpoint. ODP explicitly states that correspondences do not form part of any viewpoint. In addition,

ODP defines the concept of viewpoint *on a system*, whilst correspondences are defined *between two viewpoints*. However, we do agree that correspondences should be represented by something more powerful than UML abstraction dependencies for the reasons stated above: correspondences may require to store more information than a single UML abstraction dependency can convey, and they may be required for other purposes—e.g., for enforcing and propagating changes from one view to another.

The fact that change propagations can be considered particular cases of model transformations suggests the use of QVT as the perfect solution to the problem of representing ODP correspondences. The use of relations was initially indicated by [14] for relating concepts from different viewpoint at the metalevel but not explored any further for relating instances, which is essential for establishing proper correspondences.

RM-ODP itself explicitly states that correspondences can be used to define transformations between viewpoint elements to implement consistency checks: "One form of consistency involves a set of correspondence rules to steer a transformation from one language to another. Thus given a specification S_1 in viewpoint language L_1 and specification S_2 in viewpoint language L_2, where S_1 and S_2 both specify the same system, a transformation T can be applied to S_1 resulting in a new specification $T(S_1)$ in viewpoint language L_2 which can be compared directly to S_2 to check, for example, for behavioral compatibility between allegedly equivalent objects or configurations of objects." [8]

3 QVT

3.1 QVT Relations

MOF QVT (Query/View/Transformation) [7] is the OMG's standard for specifying MOF model queries, views and transformations. It is expected to play a central role in the Model Driven Architecture [15]. QVT defines three different (but closely related) languages for specifying transformations using declarative and imperative styles. Black-box implementations of operations can also be used to allow reuse of existing algorithms or domain specific libraries in certain model transformations.

QVT Relations is a language to write declarative specifications of the relationships between MOF models. The QVT Relations language supports object pattern matching, and implicitly creates trace classes and their instances to record what occurred during a transformation execution. Relations can assert that other relations also hold between particular model elements matched by their patterns.

QVT Relations allow for the following execution scenarios [7]:

- Check-only transformations to verify that models are related in a specified way.
- Single direction and bi-directional transformations.
- The ability to establish relationships between pre-existing models, whether developed manually, or through some other tool or mechanism.
- Incremental updates (in any direction) when one related model is changed after an initial execution.
- The ability to create as well as delete objects and values, while also being able to specify which objects and values must not be modified.

3.2 QVT Transformations

In the relations language, a transformation between candidate models is specified as a set of relations that must hold for the transformation to be successful. A *candidate model* is any model that conforms to a *model type*, which is a specification of what kind of model elements any conforming model can have. An example is:

```
modeltype EL uses "odp.UML4ODP.EL_UMLProfile"
modeltype IL uses "odp.UML4ODP.IL_UMLProfile"
transformation EVtoIV (ev : EL, iv : IL) {
      top relation EVrole2IVobjectType {...}
      top relation EVobject2IVobject {...}
      ...
}
```

Relations in a transformation declare constraints that must be satisfied by the elements of the candidate models, and specify a relationship that must hold between the elements of the candidate models. Top level relations are those that need to hold for a transformation to be successfully executed.

A relation is defined by two or more domains and a pair of when and where predicates. For instance, the following relation EVrole2IVobjectType establishes a relationship between roles in the EV specification and object types in the IV specification, whereby every enterprise role is related to one information object type with the same name (but not necessarily vice-versa, i.e., not every information object type should correspond to an enterprise role).

```
relation EVrole2IVobjectType { /* maps e-roles to i-objectTypes */
   domain ev er:Class {name=r}
   domain iv iot:Class {name=r}
   when { er.stereotypedBy("EV_Role") }
   where { er.stereotypedBy("EV_Role") and iot.stereotypedBy("IV_ObjectType") }
}
```

More precisely, relation EVrole2IVobjectType checks that for each role in the EV specification (i.e., a class stereotyped EV_Role) there is an object type with the same name in the IV specification (i.e., a class stereotyped IV_ObjectType).

A transformation can be invoked either to check two models for consistency or to modify one model to enforce consistency. In the first case, the transformation checks whether the relations hold in all directions, and report errors when they do not hold. In case of enforcement, one model acts as source and the other as target; the execution of the transformation proceeds by first checking whether the relations hold, and for relations for which the check fails, attempting to make the relations hold by creating, deleting or modifying only the target model, thus enforcing the relationship.

QVT transformations can also be used for propagating changes from one model to other. As mentioned in the QVT standard [7], "the effect of propagating a change from a source model to a target model is semantically equivalent to executing the entire trans-

formation afresh in the direction of the target model. The semantics of object creation and deletion guarantee that only the required parts of the target model are affected by the change. Firstly, the semantics of check-before-enforce ensures that target model elements that satisfy the relations are not touched. Secondly, key-based object selection ensures that existing objects are updated where applicable. Thirdly, deletion semantics ensures that an object is deleted only when no other rule requires it to exist."

4 Modeling ODP Correspondences

We have seen how QVT transformations can be specified to define general relationships between elements of two ODP viewpoint specifications (e.g, between enterprise roles and information object types, or between enterprise objects and information objects). However, these kinds of correspondences are not very common in the specification of any ODP system. Usually, correspondences are defined between particular elements of the specification (e.g., between particular objects, types, templates, or actions).

For instance, suppose that we have an ODP specification of a Banking system, in which bank accounts are modeled in the computational viewpoint as objects that support a couple of interfaces for accessing their services. In the engineering viewpoint specification, we want each of these computational objects to correspond exactly to two basic engineering objects that support the same interfaces (plus possibly other interfaces only relevant to the engineering objects concerned). The specification of such part of the system at the object template level, and using the UML profiles defined in UML4ODP, is shown in Figure 1.

In order to represent such a correspondence, we could use a set of UML abstraction dependencies between the related elements. However, this could be done in a more precise and effective way using QVT.

At the object level, we need to define a relation that establishes a correspondence between a computational object which is an instance of an Account object template, and two engineering objects that represent it in the engineering specification:

```
relation cv-account2twonv-accounts {
  domain cv a:InstanceSpecification {name=n, classifier = "Account"}
  domain nv a1:InstanceSpecification {name=n + '1', classifier = "Account1"}
  domain nv a2:InstanceSpecification {name=n + '2', classifier = "Account2"}
  when { a.stereotypedBy("CV_Object") }
  where { a.stereotypedBy("CV_Object") and a1.stereotypedBy("NV_BEO") and
    a2.stereotypedBy("NV_BEO") and DuplTemplates(a.classifier,a1.classifier,a2.classifier)
  }
}
```

We can see how it establishes that if there exists a UML InstanceSpecification stereotyped CV_Object, whose classifier is an Account, then there should be two UML InstanceSpecifications stereotyped NV_BEO, whose classifiers are Account1 and Account2, respectively. In addition, a relation called DuplTemplates should also hold between the classifiers of all these instance specifications. Such a QVT relation is pre-

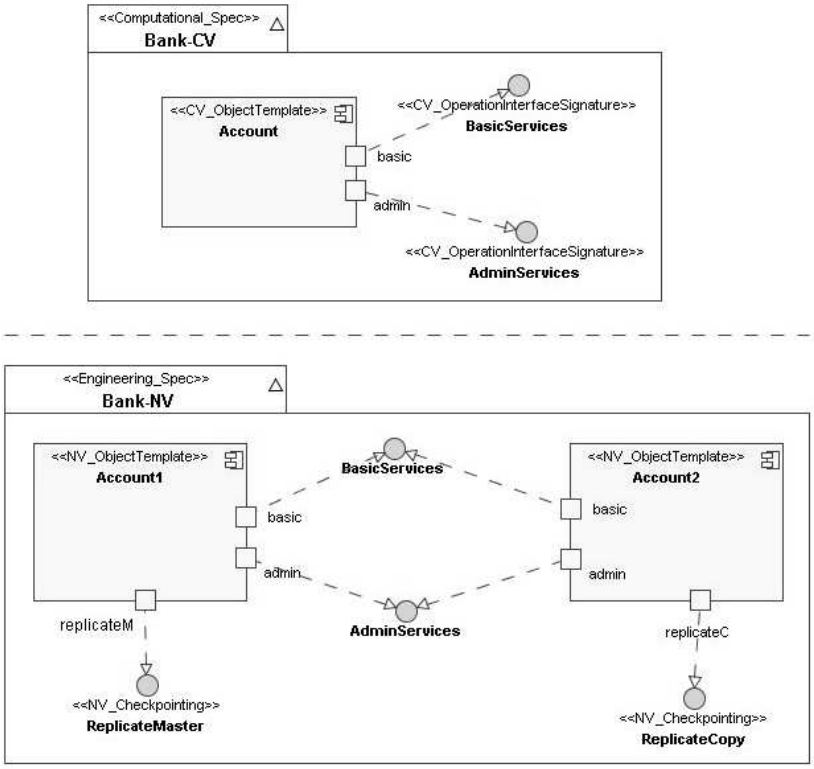


Fig. 1. Bank Account comp. objects and interfaces should be related to the corresponding eng. objects and interfaces.

cisely the one that establishes the correspondence between the appropriate computational object templates (Fig. 1):

```
relation DuplTemplates{
  domain cv a:Component {name=n}
  domain nv a1:Component {name=n + '1'}
  domain nv a2:Component {name=n + '2'}
  when { a.stereotypedBy("CV_ObjectTemplate") }
  where { a.stereotypedBy("CV_ObjectTemplate") and
    a1.stereotypedBy("NV_ObjectTemplate") and
    a2.stereotypedBy("NV_ObjectTemplate") and
    sameODPInterfaces(a,a1) and sameODPInterfaces(a,a2)
  }
}
```

This relation establishes that a given computational object template should be related to two engineering object templates (whose names should be the same, but suffixed with ‘1’ and ‘2’), and that the ODP interfaces of the computational object tem-

plate should be supported by the corresponding interfaces of the engineering object templates—as stated by the ODP required correspondence **CN-3**. This required correspondence is expressed using the sameODPInterfaces relation, that checks that every interface defined for a computational object template is supported by an interface of a given engineering object template. In the UML4ODP context, both computational and engineering object templates are modeled using UML components, and both computational and engineering interfaces are represented by UML ports. Thus, the QVT relation checks that every port of the UML component representing the computational object template has an associated port with the same name in the given UML component representing the basic engineering object template, and that the set of provided and required interfaces of each port are the same in the two specifications.

```
relation sameODPInterfaces {
  domain cv cot:Component {}
  domain nv eot:Component {}
  when {
   cot.stereotypedBy("CV_ObjectTemplate") and eot.stereotypedBy("NV_ObjectTemplate")
  }
  where { eot.ownedPort.name->includes(cot.ownedPort.name)
    and cot.ownedPort->forAll(p | p.required =
            eot.ownedPort->select(name=p.name).required)
    and cot.ownedPort->forAll(p | p.provided =
            eot.ownedPort->select(name=p.name).provided)
  }
}
```

This last relation can be reused as-is in other QVT relations to enforce the required correspondence, **CN-3**, in other ODP correspondence statements.

5 Issues for Discussion

Once we have briefly seen how QVT could be used to represent both ODP correspondence statements and ODP required correspondences, let us discuss in this section some issues that may require further investigation.

5.1 Bi-directionality and Cardinality of Correspondences

The RM-ODP is silent about the possible bi-directionality of the ODP correspondences. However, we believe such correspondences must be bidirectional so it is possible to navigate from any of the two views to the other. The idea is to be able to trace elements, i.e., given an element of a viewpoint, find all the elements in the rest of the viewpoints which are related to it (objects, policies, rules, actions, etc.).

In addition, RM-ODP seems to define correspondences just between pairs of viewpoints. However, sometimes correspondences between one and more viewpoints might be required, i.e., between one element in one viewpoint and several elements in other

viewpoints. Defining this kind of 1-M correspondences is possible with QVT relationships, although something not defined in RM-ODP.

5.2 Transitivity of Correspondences

The QVT relations presented here can be used for change propagation. This occurs when a change happens in one of the viewpoint specifications, and we want to propagate the change to all related elements in the rest of the viewpoint specifications. In this case we can consider QVT relations as model transformations, enforcing the relationships on the target models as mentioned earlier. However, this may raise some redundancy or duplication issues due to transitivity of the relations.

Suppose elements α, β and γ in viewpoints A, B and C respectively, related as follows: α is related with β and γ, and β is related with γ. How to deal with the potential redundancy that may happen when a change in element α is propagated to γ both directly from α to γ, and indirectly through β? There are cases where this does not imply any problem, as it happens when the relations just check that the elements have the same name, and we change the name of α. However, what happens when the relations add something to the elements' structure or behavior? E.g., suppose they add a suffix to the name of the element? Will we end up with a duplicated suffix in the name of γ?

Please notice how this is an example that could justify the need for establishing N-M correspondences between viewpoints.

5.3 Full Consistency of Specifications

In order to check the consistency of the specifications, we can use the ODP correspondences if we consider them as model transformations, as mentioned in the RM-ODP standards. However, complete consistency between viewpoint specifications cannot be guaranteed by ODP correspondences only. Analysis of consistency depends on the application of specific consistency techniques, most of which are based on checks for particular kinds of inconsistency, and thus cannot prove complete consistency.

This latter issue has been addressed by several people, from different perspectives. The interested reader can consult, e.g., the works by Derrick, Bowman et al. [10], the interesting book [1], and also the recent and complete work done by Remco Dijkman in his PhD thesis [11]. How to combine the use of model-driven techniques and QVT in those contexts is something we would like to explore further as part of future research.

6 Conclusions

In this paper we have sketched how QVT relations can be used to represent ODP correspondences in the context of the UML4ODP project, in an initial attempt to show that this approach is feasible. QVT relations provide more powerful mechanisms than those provided by plain OCL or UML abstraction dependencies for relating elements in different ODP viewpoints, can be modularly and independently specified, be reused to build more powerful QVT transformations, and serve both for checking the correspondences and for enforcing them.

There are still several issues open for investigation. Apart from the questions mentioned above, it is not clear whether this method is better or not than the other ones discussed here, e.g., the one proposed by Remco Dijkman [11], or by Yahiaoui et al. [12, 13]. Furthermore, apart from specifying the correspondences, can the QVT relations provide any other advantages? Can they be used, for instance, to reason about the system specifications and their consistency? And if so, how this can be achieved? Which is the underlying logic in which the reasoning can be done? Apart from consistency, what other properties can be proved from the QVT specifications of the correspondences? These are interesting questions, some of them we plan to address in a near future.

Acknowledgements

The authors would like to thank the anonymous referees for their constructive comments and suggestions. This work has been partially supported by Spanish Research Project TIN2005-09405-C02-01.

References

1. Große-Rhode, M.: Semantic Integration of Heterogeneous Software Specifications. Springer-Verlag, Berlin (2004)
2. IEEE Std. 1471: Recommened Practice for Architectural Description of Software-Intensive Systems. IEEE Standards Association. (2000)
3. ISO/IEC 10746-1 to 10746-4, ITU-T X.901 to X.904: RM-ODP. Reference Model for Open Distributed Processing. ISO & ITU-T. (1997)
4. Linington, P.: RM-ODP: The architecture. In Milosevic, K., Armstrong, L., eds.: Open Distributed Processing II, Chapman & Hall (1995) 15–33
5. ISO/IEC 10746-4, ITU-T Rec. X.904: Information technology – Open distributed processing – Reference model: Architectural Semantics. ISO & ITU-T. (1998)
6. ISO/IEC CD 19793, ITU-T Rec. X.906: Information technology – Open distributed processing – Use of UML for ODP system specifications. ISO & ITU-T. (2005)
7. OMG: MOF QVT Final Adopted Specification. Object Management Group. (2005) OMG doc. ptc/05-11-01.
8. ISO/IEC 10746-3, ITU-T Rec. X.903: Information technology – Open distributed processing – Reference model: Architecture. ISO & ITU-T. (1996)
9. ISO/IEC 15414, ITU-T Rec. X.911: Information technology – Open distributed processing – Reference model – Enterprise language. ISO & ITU-T. (2006)
10. Boiten, E.A., Bowman, H., Derrick, J., Linington, P., Steen, M.W.: Viewpoint consistency in ODP. Computer Networks **34** (2000) 503–537
11. Dijkman, R.: Consistency in Multi-Viewpoint Architectural Design. PhD thesis, University of Twente (2006)
12. Yahiaoui, N., Traverson, B., Levy, N.: Adaptation management in multi-view systems. In: Proc. of WCAT'05, Glasgow, Scotland, UK (2005) 99–105
13. Yahiaoui, N., Traverson, B., Levy, N.: A new viewpoint for change management in RM-ODP systems. In: Proc. of WODPEC 2005, Enschede, The Netherlands (2005) 1–6
14. Akehurst, D.H.: Proposal for a model driven approach to creating a tool to support the RM-ODP. In: Proc. of WODPEC 2004, Monterey, California (2004) 65–68
15. OMG: Model Driven Architecture. A Technical Perspective. Object Management Group. (2001) OMG doc. ab/2001-01-01.

Model Quality in the Context of Model-Driven Development

Ida Solheim and Tor Neple

SINTEF ICT, P.O. Box 124 Blindern, N-0314 Oslo, Norway
{ida.solheim, tor.neple}@sintef.no

Abstract. Model-Driven Development (MDD) poses new quality requirements to models. This paper presents these requirements by specializing a generic framework for model quality. Of particular interest are *transformability* and *maintainability*, two main quality criteria for models to be used in MDD. These two are decomposed into quality criteria that can be measured and evaluated. Another pertinent discussion item is the positive implication of MDD-related tools, both on the models in particular and on the success of the MDD process.

1 Introduction

1.1 Characteristics of Model-Driven Development

Model-driven development (MDD) has been around for some years, helping system engineers to analyze and document the systems to be created and maintained, and to generate parts of the program code automatically. In MDD, models are the prime artefacts. That means, models are in use throughout the whole production chain, from the early capture of user requirements to the production of executable code. Model transformations are essential, and these should preferably be automated. Indeed, tool support is by many considered a prerequisite for successful MDD (e.g. [1]).

Although MDD has been practiced for years, it did not gain ground until the Object Management Group (OMG) launched its Model-Driven Architecture (MDA™) initiative. Being "an approach to using models in software development" [2], MDA has boosted the development of tools and thereby (semi)automation of program development and maintenance. MDA motivates system development with the following characteristics:

- Many activities have models as input, or output, or both.
- Several of these activities are model transformations (while others are model analysis, model verification etc.).
- A transformation takes one or several models as input and produces a model (or models), or text, as output. During transformation, output models are supplied with domain-related information not present in the input model. An example of such a domain is the *platform* concept, often used for "implementation platform".

1.2 Model Quality – A Less Mentioned Concern

The authors of this paper believe that successful adoption of MDD depends on high-quality models, high-quality transformations, and high-quality transformation languages and tools.

While other authors have contributed to the understanding of quality related to transformations (e.g. [3]) and transformation languages (e.g. [4]), the quality of models in MDD has so far been a less mentioned concern.

According to Selic [5], *accuracy* has been the greatest problem for successful adoption of MDD. Lack of accuracy means imprecise models or modelling languages, paired with unclear rules for mapping to underlying implementation technologies.

The authors of this paper agree that Selic has a good point. However, in [5] the term accuracy is used for a collection of several undefined quality criteria. The purpose of this paper is to define more precise quality criteria for models to be used in MDD, and suggest how these criteria may be measured and evaluated.

1.3 The Structure of this Paper

The starting point for this work is a generic quality framework (chapter 2), which is specialized to a quality framework for MDD models and their environments (chapter 3). The implications of tools are discussed in chapter 4, and a conclusive summary is given in chapter 5.

2 A Generic Quality Framework

Krogstie and Sølvberg [6] presents a generic framework for discussing the quality of models. This framework will be used as a reference frame for discussing model quality in an MDD context, and will be refined for this purpose. Figure 1 depicts the framework's building blocks and their interrelationships, as described by Krogstie [7]. The explanation of the building blocks is rendered from [7] (mostly quoted):

- ***G,*** the (normally organizationally motivated) goals of the modelling task
- ***L***, the language extension, i.e., the set of all statements that are possible to make according to the graphemes, vocabulary, and syntax of the modelling languages used
- ***M***, the externalized model, i.e., the set of all statements in someone's model of part of the perceived reality written in a language

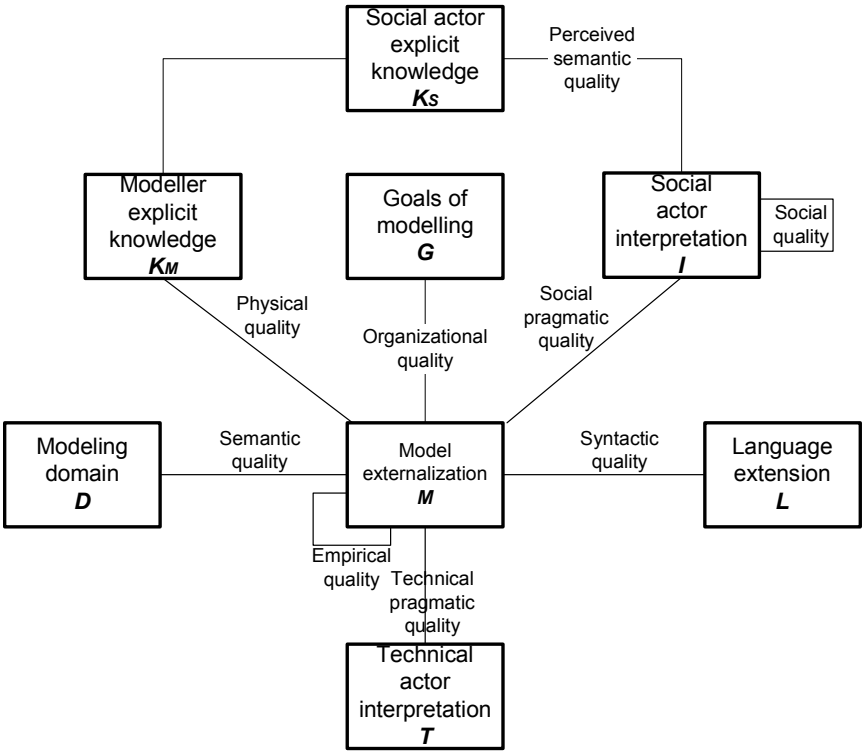


Fig. 1. Krogstie's generic framework for discussing the quality of models (rendered by courtesy of the author).

- ***D***, the domain, i.e., the set of all statements which can be stated about the situation at hand. Enterprise domains are socially constructed, and are more or less inter-subjectively agreed. That the world is socially constructed does not make it any less important to model that world.
- K_s, the relevant explicit knowledge of the set of stakeholders involved in modelling
- K_M, the relevant explicit knowledge of the set of stakeholders *actively* involved in modelling
- ***I***, the social actor interpretation, i.e., the set of all statements which the audience think that an externalized model consists of
- ***T***, the technical actor interpretation, i.e., the statements in the model as 'interpreted' by different model activators (e.g., modelling tools, transformation tools)

The various qualities are expressed as relations between pairs of these building blocks. The next chapter elaborates on model quality aspects related to MDD, refining the above framework accordingly.

3 Quality Criteria for MDD Models and their Environments

3.1 Overview

A quality framework specialized with respect to MDD is depicted in Figure 2. The authors of this paper want to emphasize *transformability* and *maintainability* as the two main quality criteria for models to be used in MDD. Models must have the ability to be transformed – to other models of greater detail (specialization), and at last to executable pieces of code for selected technical platforms. Transformability may be decomposed into:

- completeness (semantic quality)
- relevance (technical pragmatic quality)
- precision (technical pragmatic quality)
- well-formedness (syntactic quality)

Also, models for use in MDD need to be maintained during the system's lifetime. One of MDD's strengths is rapid iterations of the development cycle analysis—design—implementation—test, a feature that supports incremental development strategies. Given this setting, it is of paramount importance that changes made to the requirements are rendered correctly in the models and reflected in the code. A means to keep track of changes is to trace them, from the requirements through the necessary steps all the way to the code, and back. Therefore, maintainability of models may be decomposed into:

- traceability (technical pragmatic quality)
- well-designedness (syntactic quality)

Out of the six quality criteria listed above, only one (completeness) is explicitly mentioned in Krogstie and Sølvberg [6]. The remaining five may be considered refinements of generic relations shown in Fig. 1. The transformability and maintainability criteria are explained in the following subsections.

The *environments* of MDD models are here defined to be the change traces, the tools, and the MDD process itself. The change traces and the tools belong to the technical pragmatic quality.

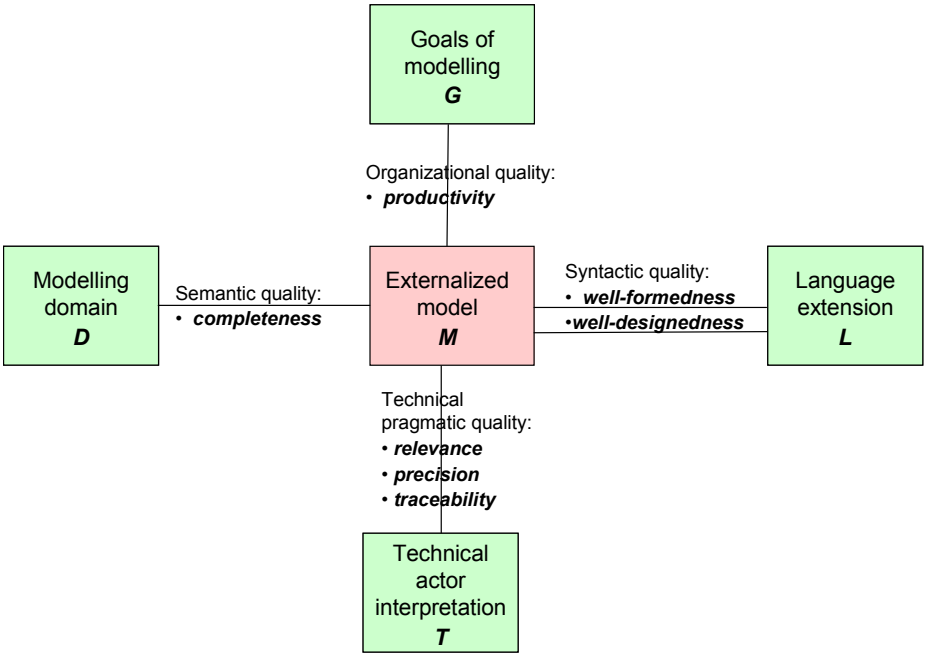


Fig. 2. A specialized framework for model quality in MDD.

Concerning the MDD process as such, we may identify a primary goal of achieving higher productivity in the development process. Hence,

- productivity (organizational quality)

may be considered a quality criterion. In accordance with [6], this is in Fig. 2 expressed as a relation between the goals of modelling (G) and the externalized model (M). However, for MDD, productivity should rather appear as a quality of M, L, T and D in combination. Productivity is hard to measure, and results cannot easily be generalized. The MODELWARE project [8] of the EU IST programme aims at measuring the productivity of MDD in industrial trials, based on approaches described in [9].

3.2 Transformability

Completeness is pointed out by Krogstie and Sølvberg [6] as an essential for the semantic quality of models. Completeness assures that the model contains all statements that are correct and relevant about the domain, and can be measured by a percentage as prescribed in [6].

Whereas Krogstie and Sølvberg [op. cit.] consider *relevance* to be a property of completeness, the authors of this paper would like to emphasize relevance as a distinct quality criterion. However, the relevance of a model used in MDD depends on

both the model itself (M) and its transformation as specified by the technical actor interpretation (T). High relevance means that no more statements are included in the model than those which are going to be transformed. Relevance can be measured as the percentage of model elements actually used in a particular transformation. Making a larger model than necessary has a negative consequence in MDD; one has to drag along unused model elements (or code), which may complicate documentation, blur comprehension and hamper maintenance.

Precision reflects the level of detail and accuracy required for a model to be transformed successfully. The result of the transformation may be another model, which in case must be well-formed. Or, the result may be program code which can be compiled without errors and which constitutes some meaningful result, e.g. a component, a class structure or an interface. It may be possible to measure precision on a scale (ordinal or interval). However, these authors prefer to evaluate model precision as yes/no. This means, either the model is sufficiently precise for transformation, or it is not.

Well-formedness is a syntactic quality of utmost importance to model transformation. According to OMG [2], a transformation from one model to another is dependent on a mapping between the two respective metamodels. Hence, any model to be transformed must comply with its metamodel. For example, a model written in UML must comply with UML's metamodel. Also, there may exist sub-languages with limitations on the vocabulary and/or grammar rules of the overall language. Examples of such sub-languages are UML profiles. A well-formed model complies not only to its metamodel, but also to its sub-language (profile) if appropriate. A measure of well-formedness should yield 100 % before transformation is started.

3.3 Maintainability

3.3.1 Traceability

Traceability has been pointed out as an important aspect of MDD. One of the purposes of maintaining traces between model elements is to check a model element's origin, e.g. in a requirement model, and to follow a model element through transformations. In the latter case, the trace can also tell what kind of transformation was used, and which transformation rule was applied. Albeit traceability doubtlessly may involve more than one model, and indeed may involve artefacts other than models, this section discusses traceability as a quality of a model. This means, to what degree the model is usable in a scenario where traceability is needed.

Traceability may be vital for the management of large MDD projects, and for the maintenance of systems built according to MDD. Tool-supported traceability may range from "enterprise-wide" traceability solutions to simple traces maintained by the modelling workbench. A model's traceability depends on unique identifiers for the different elements that constitute the model; otherwise no traces can be established. Unique identifiers are supported by some modelling tools, but not all. In addition to the identification of model elements, one will need a mechanism that logs and documents all transitions undergone by each model element. Such a mechanism is currently under development in the IST project MODELWARE [8].

A traceability metric for a model could be the model's *trace coverage*, defined as a percentage denoting the proportion of traceable model elements relative to the total number of model elements.

3.3.2 Well-designedness

The maintainability of object-oriented systems has been studied by several authors, e.g. Briand [10]. The main approach has been various combinations of measurements, obtained by counting properties of object-oriented structures found in class diagrams. Marinescu [11] introduced a quality model for object-oriented systems, applying well-known metrics for the purpose of revealing particular design flaws. Among the design flaws that can be revealed by his method, are flaws resulting from *not* using selected design patterns described by Gamma et al. [12].

Well-designed models are understandable and tidy. In MDD, well-designedness deserves much attention because the models are the prime artefacts. Maintenance should preferably start with the models resulting from the last development cycle. If changes are made directly to the generated code, they should be reflected in the models as soon as possible to ensure the correspondence between the models and the code. Bad model design may complicate the code, confuse the developers, ruin the model-code correspondence and impede the use of MDD.

4 The Implications of Tools

In MDD, tools are used to create models, to transform one model into another, to generate non-model software artefacts, to maintain traces, etc. In such a setting, the human model-creation steps can be heavily guided by the tools. This means that several quality parameters can be kept at sufficient levels through guides and constraints in the tooling. It is also probable that the modeller will put most work into those models that will be subject to usage further down the MDD transformation chain.

A modelling tool will typically not allow a model to violate its metamodel. At least, the model will be compliant with *the tool's interpretation* of the metamodel. This is a feature that has been observed in UML tools in the past, when tool vendors have added capabilities not compliant to the UML metamodel as defined by the standard. Such extensions may cause problems in an MDD tool chain if a common non-standard metamodel, shared between the tools, is required.

A positive feature of some UML modelling tools is a mechanism allowing the user to check whether a model is compliant with the applied profile. In MDD, this is essential as most UML model transformations use stereotypes and extra properties in the transformation process. While the profile provides explicit language constraints, the tool enforces these constraints on the models. The quality of tool support for profile adherence is thus shared between the profile itself (how explicit are the constraints) and the tool (how well are these constraints enforced). In these cases, the quality of the model at hand is therefore a combination of the quality of the model and the quality of the applied profile.

Modelling tools can also help ensure that the structure (e.g. package organisation) of a model is in accordance with the expectations of the down-chain tools. This is

typically done by the use of model templates or more formally defined constraints on the model structure.

5 Conclusion

Models have been used for years without direct influence on system implementation. However, the adoption of MDD forces system developers to spend more effort on making high-quality models. This paper has presented a framework for reasoning about model quality in the context of MDD. Since (automatic) model transformation is a crucial activity in MDD, several quality measures depend on both the model and the transformation (or transformation tool). Such dependency is indicated by the association line between M and T in Fig. 2. Although measures may be obtained on an ordinal or ratio scale, some quality criteria need to reach a sufficiently high level – a threshold – in order for transformations to succeed. The table below gives a summary of the quality criteria and suggestions of how to measure and evaluate them.

Quality Criterion	Type of Quality	Explanation
Transformability		
Completeness	semantic	The model contains all statements that are correct and relevant about the domain (from [6]). Suggested measurement unit: percentage.
Well-formedness	syntactic	The model complies with its metamodel, and also with its specified language profile, if appropriate. Suggested measurement unit: percentage. Suggested evaluation: yes/no.
Precision	technical pragmatic	The model is sufficiently accurate and detailed for a particular automatic transformation. Suggested evaluation: yes/no.
Relevance	technical pragmatic	The model contains only the statements necessary for a particular transformation. Suggested measurement unit: percentage.
Maintainability		
Traceability	technical pragmatic	The model's elements can be traced backward to their origin (requirements), and forward to their result (another model or program code). Suggested metric: *trace coverage,* the proportion of traceable model elements relative to the total number of model elements.
Well-designedness	syntactic	The model has a tidy design, making it understandable by humans and transformable to an understandable and tidy result. Suggested metric: The quality model of Marinescu [11], preferably extended with other diagrams than class diagrams.

The use of tools in MDD serves several purposes. In addition to facilitating the drawing, maintenance and transformation of models, tools also have some built-in quality controls. It is desirable that the quality controls performed by tools are extended to support as many as possible of the quality criteria listed above.

Future work will apply the presented quality framework to models used in MDD projects within industry or public administration. Such trials are expected to give valuable feedback to the appropriateness and further refinement of the framework.

Acknowledgements

This work has been conducted in the context of MODELWARE, a project co-funded by the European Commission under the "Information Society Technologies" Sixth Framework Programme (2002-2006). Information included in this document reflects only the author's views. The European Community is not liable for any use that may be made of the information contained therein.

References

1. Alanen, M., et al.: *Model Driven Engineering: A Position Paper*. 1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES'04 (2004)
2. Object Management Group: MDA Guide. Ver. 1.0.1. http://www.omg.org/docs/omg/03-06-01.pdf (2003).
3. Bézivin, J., et al.: *The ATL Transformation-based Model Management Framework*. IRIN, Université de Nantes (2003)
4. Grønmo, R., et al.: *Evaluation of the Proposed QVTMerge Language for Model Transformations*. The First International Workshop on Model-Driven Enterprise Information Systems (MDEIS-2005). Miami (2005)
5. Selic, B., The Pragmatics of Model-Driven Development. In: *IEEE Software*, September/October 2003, http://computer.org/software
6. Krogstie, J. and Sølvberg, A.: *Information systems engineering - Conceptual modeling in a quality perspective*. Kompendiumforlaget. Trondheim, Norway (2003)
7. Krogstie, J. (2003). Evaluating UML Using a Generic Quality Framework. In: Favre, L. (ed.): UML and the Unified Process. IRM Press 1-22.
8. IST Project 511731 (2004-2006). *MODELWARE. Modeling solution for software systems,* http://www.modelware-ist.org/.
9. MODELWARE: *MDD Business Metrics (in prep.)*. Sixth Framework programme (2006)
10. Briand, L., Daly, J., and Wüst, J.: A Unified Framework for Coupling Measurement in Object-Oriented Systems. In: *IEEE Transactions on Software Engineering,* Vol. 25(1), No. January/February, 1999
11. Marinescu, R.: *Measurement and Quality in Object-Oriented Design*. University of Timisoara (2002)
12. Gamma, R., et al.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley (1994)

Transformations

Model-Based Development with Validated Model Transformation

László Lengyel, Tihamér Levendovszky, Gergely Mezei and Hassan Charaf

Budapest University of Technology and Economics,
Goldmann György tér 3.
1111 Budapest, Hungary
{lengyel, tihamer, gmezei, hassan}@aut.bme.hu

Abstract. Model-Driven Architecture (MDA) as a model-based approach to software development facilitates the synthesis of application programs from models created using customized, domain-specific model processors. MDA model compilers can be realized by graph rewriting-based model transformation. In Visual Modeling and Transformation System (VMTS), metamodel-based transformation steps enables assigning OCL constraints to model transformation steps. Based on this facility, the paper proposes a novel validated model transformation approach that can ensure to validate not only the individual transformation steps, but the whole transformations as well. The discussed approach provides a visual control flow language to define transformations visually in a simple way that results more efficient development process. The presented methods are illustrated using a case study from the field of model-based development.

1 Introduction

Model-driven development approaches (e.g. Model-Integrated Computing (MIC) [1] and OMG's Model-Driven Architecture (MDA) [2] emphasize the use of models at all stages of system development. They have placed model-based approaches to software development into focus.

MIC advocates the use of domain-specific concepts to represent the system design. Domain-specific models are then used to synthesize executable systems, perform analysis or drive simulations. Using domain concepts to represent the system design helps increase productivity, makes systems easier to maintain, and shortens the development cycle.

MDA offers a standardized framework to separate the essential, platform-independent information from the platform-dependent constructs and assumptions. A complete MDA application consists of a definitive platform-independent model (PIM), one or more platform-specific models (PSM) including complete implementations, one on each platform that the application developer decides to support. The platform-independent artifacts are mainly UML and other software models containing enough specification to generate the platform-dependent artifacts automatically by model compilers.

Transformations appear in many different situations in a model-based development process. A few representative examples are as follows. (i) Refining the design to implementation; this is a basic case of PIM/PSM mapping. (ii) Aspect weaving; the integration of aspect models/code into functional artifacts is a transformation on the design. (iii) Analysis and verification; analysis algorithms can be expressed as transformations on the design.

One can conclude that transformations in general play an essential role in model-based development, thus, there is a need for highly reusable model transformation tools. These tools must make the model transformation flexible and expressive, therefore, it should preferably be defined visually. Furthermore they should support control flow, constraints, parameter passing between sequential rules, and conditional branching. Moreover, they should be user friendly and simple to use to make the development as efficient as it is possible.

The approach presented here uses graph rewriting-based visual model transformation. To define the transformation steps precisely and support the validated model transformation beyond the structure of the visual models, additional constraints must be specified which ensure the correctness of the attributes, or other properties can be enforced. Using Object Constraint Language (OCL) [3] constraints provides a solution for these issues. The use of OCL as a constraint and query language in modeling is found to be simple and powerful. We have shown that it can be applied to model transformations as well [4].

The main contribution of the current paper is the validated online model transformation. Section 2 presents the motivation on a real word case study. Section 3 introduces the principles of the validated model transformation: the relation between the pre- and postconditions and OCL constraints propagated to model transformation steps. Section 3.1 shortly presents the Visual Control Flow Language (VCFL) of the Visual Modeling and Transformation System (VMTS) [5] that facilitates an efficient and simple way to define model transformations visually. Using the motivation case study, Section 3.2 discusses the details of the validated model transformation. The approach presented here makes possible to require transformation steps as well as the whole transformations to validate, preserve or guarantee certain properties during the transformation. Section 4 summarizes the related work and compares VMTS with other model transformation approaches. Finally, conclusions are provided.

2 Motivation – A Case Study

To illustrate the motivations on a real word example a case study is provided. The case study is a variation of the “class model to relational database management system (RDBMS) model” transformation (also referred to as object-relational mapping).

The requirements stated against the transformation that it should guarantee are the following properties:

- Classes that are marked as non-abstract in the source model should be transformed into a single table of the same name in the target model. The resultant table should contain one added primary key column, one column for each at-

tribute in the class, and one or more columns for associations based on the next rule.

- In general, an association may, or may not, map to a table. It depends on the type and multiplicity of the association.
  - Many-to-many (N:N) associations, should be mapped to distinct tables. The primary keys for both related classes should become attributes of the association table (foreign keys). Foreign keys do not allow NULL values
  - One-to-many (1:N) and associations, using one or more foreign key columns should be merged into the table for the class on the "many" side.
  - For one-to-one (1:1) associations, also the foreign key should be buried optionally in one of the affected tables.
- Parent class attributes should be mapped into tables created from inherited classes.

The required rules jointly guarantee that the generated database is in third normal form [6].

At the implementation level, system validation can be achieved by testing. Various tools and methodologies have been developed to assist in testing the implementation of a system (for example, unit testing, mutation testing, and white/black box testing). However, in case of model transformation environments, it is not enough to validate that the transformation engine itself works as it is expected. The transformation specification should also be validated.

There are only few and not complete facilities provided for testing offline transformation specifications in an executable style. Related to the expected output there is nothing that can be guaranteed by these transformations. The transformation should be tested: not only the syntactical but the semantical correctness is also required. In fact, the testing requires huge efforts, and even after the testing it is not guaranteed that the transformation produces the expected output for all valid input. The reason is that there is no real possibility that the testing covers all the possible cases. But, in the case of the case study the following issues should be guaranteed by the transformation: (i) Each table has primary key, (ii) each class attribute is part of a table, (iii) each parent class attribute is part of a table created for its inherited class, (iv) each many-to-many association has a distinct table, (v) each one-to-many and one-to-one association has merged into the appropriate tables, (vi) foreign keys not allow NULL value, and (vii) each association class attribute buried into the appropriate table based on the multiplicities of its association.

There is a need for a solution that can validate model transformation specifications: online validated model transformation that guarantees if the transformation finishes successfully, the generated output (database schema) is valid, and it is in accordance with the requirements above.

3 Validated Model Transformation

Graph rewriting [7] is a powerful technique for graph transformation with a strong mathematical background. The atoms of graph transformations are rewriting rules, each rule consists of a left-hand side graph (LHS) and right-hand side graph (RHS).

Applying a graph rewriting rule means finding an isomorphic occurrence (match) of LHS in the graph the rule being applied to (host graph), and replacing this subgraph with RHS.

The Object Constraint Language is a formal language for the analysis and design of software systems. It is a subset of the UML standard [8], and OCL allows software developers to write constraints and queries over object models. A precondition to an operation is a restriction that must be true immediately prior to its execution. Similarly, a postcondition to an operation is a restriction that must be true immediately after its execution.

A *precondition* assigned to a transformation step is a *boolean* expression that must be true at the moment when the transformation step is fired. Similarly, a *postcondition* assigned to a transformation step is a *boolean* expression that must be true after the completion of a transformation step. If a *precondition* of a transformation step is not true then the transformation step fails without being fired. If a *postcondition* of a transformation step is not true after the execution of the transformation step then the transformation step fails. A direct corollary of this is that an OCL expression in LHS is a precondition to the transformation step, and an OCL expression in RHS is a postcondition to the transformation step. A transformation step can be fired if and only if all conditions enlisted in LHS are true. Also, if a transformation step finished successfully then all conditions enlisted in RHS must be true [4].

3.1 VMTS Visual Control Flow Language

VMTS is an n-layer metamodeling environment which supports editing models according to their metamodels, and allows specifying OCL constraints. Models are formalized as directed, labeled graphs. VMTS uses a simplified class diagram for its root metamodel ("visual vocabulary"). Also, VMTS is a model transformation system, which transforms models using graph rewriting techniques. Moreover, the tool facilitates the verification of the constraints specified in the transformation step during the model transformation process.

Model-to-model transformations often need to follow an algorithm that requires a stricter control over the execution sequence of the steps. The VMTS approach is a visual approach and it also uses graphical notation for control flow: stereotyped UML activity diagram [8]. VMTS Visual Control Flow Language (VCFL) is a visual language for controlled graph rewriting and transformation, which supports the following constructs: sequencing transformation steps, branching with OCL constraints, hierarchical steps, parallel execution of the steps, and iteration.

The branching construct is required, because often, the transformation that we would like to apply depends on a condition. In VCFL, OCL constraints assigned to the decision elements can choose between the paths of optional numbers, based on the properties of the actual host model and the success of the last transformation step (*SystemLastRuleSucceed*).

In VMTS, LHS and RHS of the transformation steps are built from metamodel elements. This means that an instantiation of LHS must be found in the input model instead of the isomorphic subgraph of LHS.

VMTS facilitates a refined description of the transformation steps. When the transformation is performed, the changes are specified by the RHS and *internal causality* relationships defined between the LHS and the RHS elements of a transformation step. Internal causalities can express the modification or removal of an LHS element, and the creation of an RHS element. XSLT scripts can access the attributes of the objects matched to the LHS elements, and produce a set of attributes for the RHS element to which the causality points.

The interface of the transformation steps allows the output of one step to be the input of another step (parameter passing). In VCFL, this construction is referred to as *external causality*. This feature accelerates the matching and reduces the complexity.

3.2 Validated Solution of the Case Study

In this section, a validated solution for the transformation *Class2RDBMS* supported by VCFL is presented. The case study follows the entity-driven database design and the existence-based identity implementation [6]. The metamodel for class models is shown in Fig 1a. A model consists of classes and relations between them (*Inheritance*, *Association* and *Dependency*). The *MetaClass* attributes describes the following. A class can be abstract, and it consists of *ClassAttributes* and *ClassOperations*.

The metamodel for RDBMS models is depicted in Fig. 1b. An RDBMS model consists of one or more tables. A table consists of one or more columns, which are defined as attributes of the metatype *Table*.

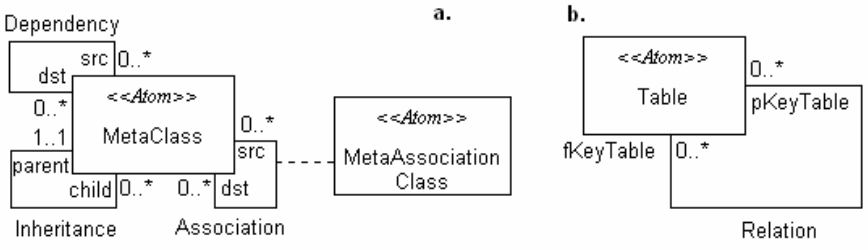


Fig. 1. VMTS Class diagram and Relational Database metamodels.

An example input and its required output model are depicted in Fig. 2. In the input model, the classes *Inhabitant* and *Institute* are abstract. The relation between the classes *Adult* and *Institute* is N:N. In Fig. 2b, there is a table for each non-abstract class and there are two connection tables for the N:N relationships (tables *Adult_School* and *Adult_Company*). Each table enlists its columns and their data type.

The control flow model of the case study (Fig. 3) can be divided into three parts according to the goal of the units. (i) The large loop on the top is responsible for the table creation and inheritance-related issues. (ii) The step *ProcessAssociation* processes the associations. (iii) Finally, the last steps remove the helper nodes and temporary associations.

One of the major challenges is to process the inheritance hierarchy properly, so the transformation must traverse the inheritance chains, because the parent class association should be taken into account recursively by subclasses.

The first step (*CreateTable*) is depicted in Fig. 4a. It matches a non-abstract class and creates a table based on it.

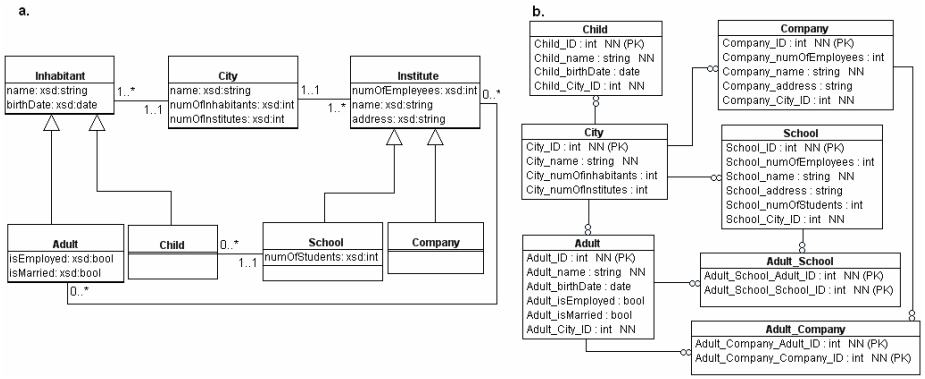


Fig. 2. (a) Example input of the case study, (b) Required output of the example input model.

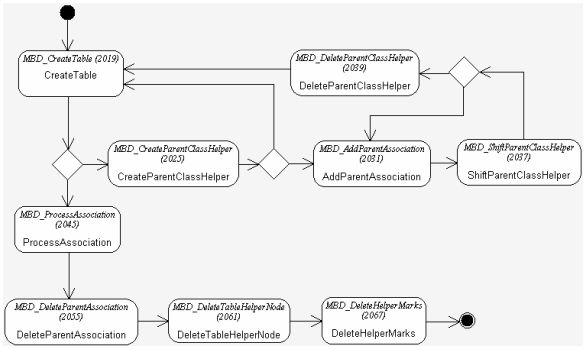


Fig. 3. The VCFL model of the transformation *Class2RDBMS*.

To require certain properties of the transformation step *CreateTable* the following constraints are applied:

```
context Class inv NonAbstract:
not self.abstract
```

The constraint *NonAbstract* is assigned to the pattern rule node (PRN) *Class* in LHS of the step *CreateTable*. This link forms a precondition, it requires the step to process only non-abstract classes.

```
context Table inv PrimaryKey:
self.columns->exists(c | c.datatype = 'int' and c.is_primary_key)
```

The constraint *PrimaryKey* is a postcondition of the step *CreateTable*, it is assigned to the PRN *Table*. This *guarantee* type constraint requires the step that all created table has a primary key of *int* type.

```
context Table inv PrimaryAndForeignKey:
not self.columns->exists(c | (c.is_primary_key or
c.is_foreign_key) and c.allows_null)
```

The constraint *PrimaryAndForeignKey* of *guarantee* type is also a postcondition that necessitates the primary and foreign key columns do not allow NULL values.

```
context Atom inv ClassAttrsAndTableCols:
self.class.attribute->forAll(self.table.column->
exists(c | (c.columnName = class.attribute.name))
```

The *guarantee* type constraint *ClassAttrsAndTableCols* is linked to the PRN *TableHelperNode*, it requires that each class attribute should have a created column with the same name in the resultant table.

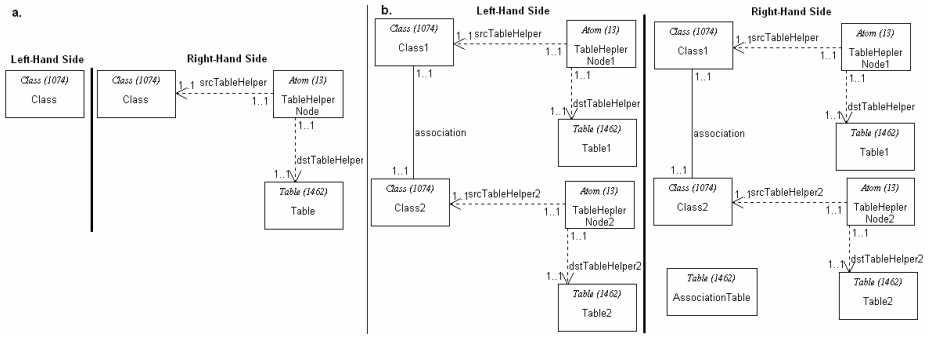


Fig. 4. Transformation steps (a) *CreateTable* and (b) *ProcessAssociation*.

If the step *CreateTable* was successful, the decision object selects the branch pointing to the step *CreateParentClassHelper*, otherwise it selects *ProcessAssociation*.

Step *AddParentAssociation* creates a temporary association that links the subclass to the neighbors of the parent class. These associations facilitate that the step *ProcessAssociations* processes not only the direct associations of a class, but the association of its parents as well.

The external causalities defined between the steps *ShiftParentClassHelper* and *AddParentAssociation* are depicted in Fig. 5. The *ParentClassHelperNode* connects a subclass with its parent class, but the parent class can also have a parent. The transformation must traverse the whole inheritance hierarchy. The step *ShiftParentClassHelper* removes the original *ParentClassHelperNode* and adds a new one which links the subclass to the parent of the parent class.

The step *ProcessAssociation* (Fig. 4b) uniformly processes the associations and the helper parent associations as well. It creates association tables (N:N associations), and

completes the already existing tables (1:N and 1:1 associations) with new foreign key columns. The following constraints are assigned to the step *ProcessAssociation*:

```
context Association inv OneToOneOrOneToMany:
(self.leftMaxMultiplicity = '1' or self.rightMaxMultiplicity = '1')
implies self.attribute->forAll (self.class1.helperNode.table.column->
exists(c | (c.columnName = attribute.name)) or self.attribute->forAll
(self.class2.helperNode.table.column->exists(c | (c.columnName = at-
tribute.name))
```

The constraint *OneToOneOrOneToMany* guarantees that the attributes of the one-to-one and the one-to-many association are buried into one of the tables created for the classes connected by the actually processed association.

```
context Association inv ManyToMany:
(self.leftMaxMultiplicity = '*' and self.rightMaxMultiplicity = '*')
implies self.attribute->forAll(self.class1.helperNode.table. con-
nectTable.column->exists(c | (c.columnName = attribute.name))
```

The constraint *ManyToMany* guarantees that, for each many-to-many type association in the resulted model, there is a distinct table. Furthermore, the table contains all attributes of the association with the same name.

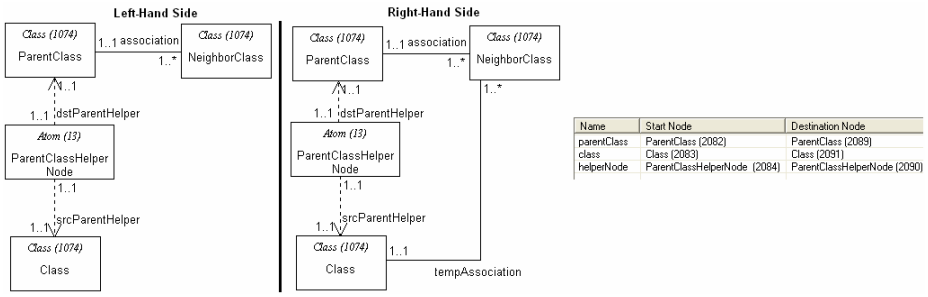


Fig. 5. Transformation step *AddParentAssociation* and external causalities between steps *ShiftParentClassHelper* and *AddParentAssociation*.

The last three transformation steps remove the remaining instances of the helper nodes, and restore the original properties of the class model elements. As a result of these steps, the input model becomes free of any helper structure.

The constraints assigned to the transformation steps guarantee the requirements from Section 2. As it is presented, after a successful step execution the conditions hold and the output is valid that cannot be achieved without constraints.

4 Related Work and Comparison

Many approaches have been introduced in the field of graph grammars and transformations to capture graph domains; for instance, the GReAT [9], the PROGRES [10],

the FUJABA [11], the VIATRA [12], and AGG [13]. These approaches are specific to the particular system, and each of them has some features that others do not offer.

The GReAT framework is a transformation system for domain specific languages (DSL) built on metamodeling and graph rewriting concepts. The control structure of GReAT allows specifying an initial context for matching to reduce the complexity of the general matching case. PROGRES is a visual programming language in the sense that it has a graph-oriented data model and a graphical syntax for its most important language constructs. In FUJABA, a combination of activity diagrams and collaboration diagrams (story-diagrams) are used to express control structures. VIATRA is a model transformation framework, its attribute transformation is performed by abstract state machine statements, and there is built-in support for attributes of basic Java types. AGG is a visual tool environment consisting of editors, interpreter and debugger for attributed graph transformation; attribute computation by Java. The control structure of AGG is given by layers.

The Model-Driven Architecture offers a standard interface to implement model transformation tools. The transformation related part of MDA is the Query, Views, Transformation for MOF 2.0 [14]. Three types of operations are provided: *queries* on models, *views* on metamodels and *transformation* on models.

Compared to other approaches, VMTS meets the expectations in model-to-model and model-to-code transformation. VMTS has state of the art mechanisms for validated model transformation, constraint management and control flow definition. It has several standalone algorithms and other solutions that make them efficient.

VMTS has a unique constraint management and online transformation validation support. It provides a high-level control flow language with several constructs that optimize and make the transformations highly configurable: external causalities, efficient branch selecting, and pivot nodes. The constraint-driven branching mechanism of the VMTS is unique in the sense that the decision is made not only based on the actual state of the input model but using system variables (*SystemLastRuleSucceed*) as well. If a transformation step fails and the next element in the control flow is a decision object, then it could provide the next branch based on the constraints. This VMTS construct accelerates and makes the transformation more efficient and the control flow model simpler, there is no need to define test rules.

5 Conclusions

Model-based development necessitates the transformation of models between different stages of the design process. These transformations must be precisely – preferably visually – specified. In this paper, a graph-transformation-based technique for specifying such a model transformation is presented. It has been shown that VMTS provides a high level visual language to define transformations in an easy way. In the provided control flow approach the transformations are represented in the form of explicitly sequenced transformation steps. We have shown the fundamental concepts of the VMTS approach, namely, the metamodel-based model transformation steps, the external- and internal-causalities for parameter passing, constraint support, and conditional branching with OCL constraints.

The main result of the paper is illustrating online validated model transformation that applying OCL constraints propagated to transformation steps facilitates to require the whole transformations to validate, preserve or guarantee certain model properties.

VCFL has already been applied in MDA-based industrial projects successfully, such as generating user interface from resource model, user interface handler code from statechart model for Symbian [15], and .NET CF mobile platforms [4].

Acknowledgements

The activities described in this paper supported, in part, by Information Technology Innovation and Knowledge Centre.

References

1. J. Sztipanovits, and G. Karsai, Model-Integrated Computing, IEEE Computer, Apr. 1997, pp. 110-112.
2. OMG MDA Guide Version 1.0.1, OMG, doc. number: omg/2003-06-01, 12th June 2003, http://www.omg.org/docs/omg/03-06-01.pdf
3. OMG Object Constraint Language Spec. (OCL), www.omg.org
4. L. Lengyel, T. Levendovszky, H. Charaf, Implementing an OCL Compiler for .NET, In Proceedings of the 3rd International Conference on .NET Technologies, Pilsen, Czech Republic, May-June 2005, pp. 121-130.
5. The VMTS Homepage. http://avalon.aut.bme.hu/~tihamer/research/vmts
6. Michael R Blaha, and William Premerlani, Object-Oriented Modeling and Design for Database Applications, Prentice Hall, 1998.
7. G. Rozenberg (ed.), Handbook on Graph Grammars and Computing by Graph Transformation: Foundations, Vol.1 World Scientific, Singapore, 1997.
8. OMG UML 2.0 Specifications, http://www.omg.org/uml/
9. G. Karsai, A. Agrawal, F. Shi, J. Sprinkle, On the Use of Graph Transformation in the Formal Specification of Model Interpreters, Journal of Universal Computer Science, 2003.
10. J. Reekers, A. Schürr, Defining and Parsing Visual Languages, Journal of Visual Languages and Computing, 8, Academic Press, 1997, pp. 27-55.
11. H. J. Köhler, U. A. Nickel, J. Niere, A. Zündorf, Integrating UML Diagrams for Production Control Systems, ICSE, Limerick Ireland, ACM Press, 2000, pp. 241-251.
12. D. Varró and A. Pataricza, "VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML", SoSyM, 2003.
13. G. Taentzer, AGG: A Graph Transformation Environment for Modeling and Validation of Software, In J. Pfaltz, M. Nagl, and B. Boehlen (eds.), Application of Graph Transformations with Industrial Relevance (AGTIVE'03), vol. 3062. Springer LNCS, 2004.
14. OMG Query/View/ Transformation. http://www.omg.org/docs/ptc/05-11-01.pdf.
15. L. Lengyel, T. Levendovszky, G. Mezei, B. Forstner, H. Charaf, Metamodel-Based Model Transformation with Aspect-Oriented Constraints, International Workshop on Graph and Model Transformation, GraMoT, ENTCS Vol. 152, Tallinn, Estonia, 2005, pp. 111-123.

Abstract Platform and Transformations for Model-Driven Service-Oriented Development

João Paulo A. Almeida[1,2], Luís Ferreira Pires[2], Marten van Sinderen[2]

[1]Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
JoaoPaulo.Almeida@telin.nl
[2]Centre for Telematics and Information Technology, University of Twente,
P.O. Box 217, 7500AE, Enschede, The Netherlands
{l.ferreirapires, m.j.sinderen}@ewi.utwente.nl

Abstract. In this paper, we discuss the use of abstract platforms and transformation for designing applications according to the principles of the service-oriented architecture. We illustrate our approach by discussing the use of the service discovery pattern at a platform-independent design level. We show how a trader service can be specified at a high-level of abstraction and incorporated in an abstract platform for service-oriented development. Designers can then build platform-independent models of applications by composing application parts with this abstract platform. Application parts can use the trader service to publish and discover service offers. We discuss how the abstract platform can be realized into two target platforms, namely Web Services (with UDDI) and CORBA (with the OMG trader).

1 Introduction

The Model-Driven Architecture (MDA) has been introduced as an approach to manage system and software complexity in distributed application design. MDA defines a set of basic concepts such as model, metamodel and transformation, and proposes a classification of models that offer different abstractions [16]. The main benefits of software development based on MDA – software stability, software quality and return on investment – stem from the possibility to derive implementations of an application in different platforms from the same platform-independent models (PIMs), and to automate to some extent the model transformation process.

Service-oriented computing (SOC) promises to deliver the methods and technologies to facilitate the development and maintenance of distributed (enterprise) applications [21]. The service-oriented paradigm is in essence characterized by the explicit identification and description of the externally observable behaviour, or service, of an application. Applications can then be discovered and linked, based on the description of their externally observable behaviour [22]. According to this paradigm, developers in principle do not need to have knowledge about the internal functioning and the technology-dependent implementation of the applications being linked. Often the term service-oriented architecture (SOA) is used to refer to the

architectural principles that underlie the communication of applications through their services [8].

We can observe from the above that service-oriented computing and model-driven engineering share some common goals, namely they both strive to facilitate development and maintenance of distributed enterprise applications, although they achieve these goals in different ways. In this paper we discuss a combination of MDA and SOA, resulting in a model-driven service-oriented development approach that can profit from the benefits of both these developments.

In particular, this paper provides the following contributions to model-driven service-oriented development:

1. we prescribe how services can be modelled in a platform-independent manner. For that, we use a general-purpose behaviour modelling language called Interaction Systems Design Language (ISDL) [13, 23] in combination with UML [19] and OCL [18];
2. we incorporate the service discovery pattern to the platform-independent design level. Our solution consists of modelling a trader service at a high-level of abstraction, and including it in an *abstract platform for service-oriented development*. This enables designers to build platform-independent models of an application by composing application parts with this abstract platform. Application parts can then use the service trader to publish and discover service offers;
3. we discuss the implementation (via transformations) of platform-independent models into two target platforms, namely Web Services [27, 28] (with UDDI repositories [15]) and CORBA (with the OMG trader [17]). We discuss how the characteristics of the abstract platform are accommodated during this transformation step.

The paper is organised as follows. Section 2 presents an overview of the different levels of models and model transformations addressed in this paper. Section 3 presents the proposed abstract platform for service-oriented development. Section 4 discusses the implications of the abstract platform for model transformations that lead to platform-specific realisations, and illustrates the approach with an application example. Finally, Section 5 summarises our results and indicates topics for future work.

2 Design Process Overview

We consider the following organization of the model-driven service-oriented development process into different levels of models: (i) the application service specification level, which describes the services offered by application parts to their environment; (ii) the platform-independent application design level and (iii) the platform-specific application design level. In this paper, we focus on the latter two levels.

The platform-independent application design level describes services that make use of an abstract platform [3, 5]. This abstract platform consists of an abstraction of service infrastructure characteristics that are assumed for the platform-independent design level. The abstract platform we discuss here supports the service discovery pattern at a platform-independent design level, and is further referred to as *SOA*

trader abstract platform in this paper. The service discovery pattern we adopt uses a trader, with which potential service consumers interact to find services based on service properties [26].

The platform-specific application design level describes the realisation of the platform-independent application design for a particular middleware platform. In order to show the flexibility of the relation between the platform-independent application design level and the platform-specific application design level two different middleware platforms are used, namely, Web Services and CORBA.

Fig. 1 depicts the organisation of the design trajectory we assume in this paper, with the three aforementioned levels of models. It reveals the composition of application services and the two elements that form the SOA trader abstract platform (in grey): the service trader and the underlying SOA abstract platform. In addition, it reveals the use of two target concrete platforms, namely Web Services and CORBA. Model transformations are depicted as arrows from a source model to a target model.

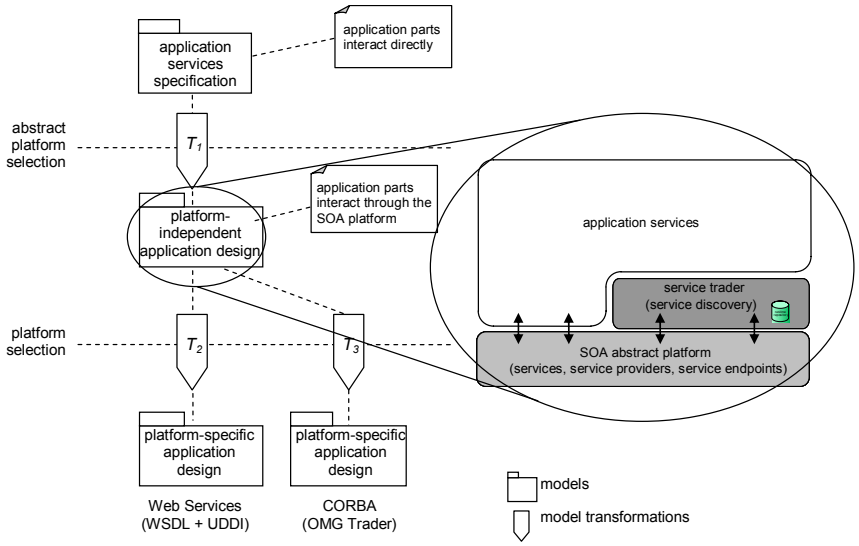


Fig. 1. Design trajectory consisting of three levels of models.

3 The SOA Trader Abstract Platform

This section defines the elements of the SOA abstract platform. We combine the two abstract platform definition approaches we have defined in [4]: the *language-level approach* and the *model-level approach*. In the language-level approach, the characteristics of an abstract platform are implied by the set of modelling constructs, patterns and styles used to model the application. For example, using "signals" in UML implies an abstract platform based on asynchronous messaging. In the model-level approach, the characteristics of an abstract platform are implied by the set of design artefacts that comprise the abstract platform. The trader service defined in this paper is an example of such a design artefact. An application designer can build the

application by composing application parts with the abstract platform. In this approach, the modelling language is used to describe: (i) the application, (ii) any design artefacts included in the abstract platform, and (iii) the composition of the application and these artefacts.

3.1 Overview

We first define the underlying *SOA abstract platform*, using a language-level approach. The language adopted for this level is ISDL [13], which is suitable for the definition of services and their interactions. This language has a formal semantics and conformance rules, which allow one to assess the conformance of behaviour refinements. The concepts in ISDL are not constrained by UML, and provide better support for the middleware-platform-independent modelling of interactions, as argued in [2]. We use UML class diagrams to model information attributes used in ISDL behavioural specifications, and OCL to model constraints on these attributes. The ISDL metamodel is defined as a MOF metamodel in [7], which facilitates its combination with UML and OCL. Fig. 2 depicts the modelling constructs of ISDL, UML and OCL schematically (language-level).

The *SOA trader abstract platform* is built on top of the underlying service-oriented abstract platform and is defined with a model-level approach. This abstract platform provides a trader service, which is defined in ISDL. Information attributes (e.g., service offers) are described with UML. The use of a trader service is a well established pattern of service discovery in service-oriented architectures. Examples of service traders in middleware platforms are the OMG CORBA trader [17] and the UDDI registry [15] (a Web Services technology). Our trader service resembles the trading function that has been defined in the scope of the Reference Model for Open Distributed Processing (RM-ODP) [14, 11].

Fig. 2 shows schematically how the elements of the SOA trader abstract platform are defined and incorporated in the platform-independent application design.

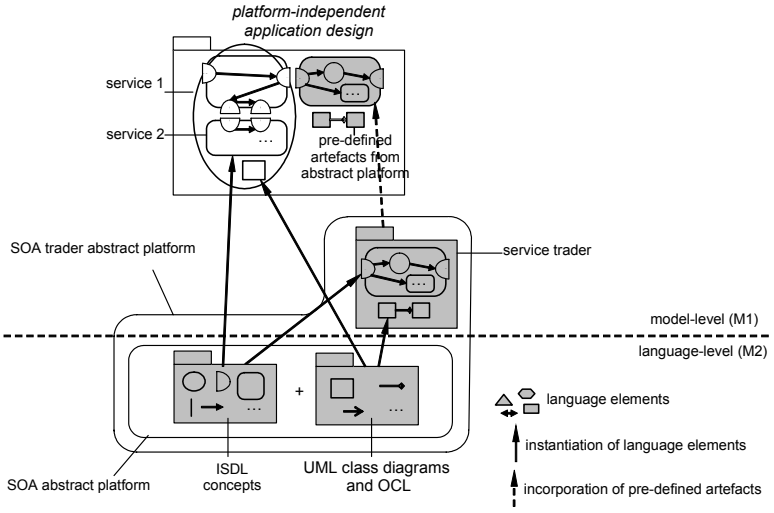


Fig. 2. SOA trader abstract platform definition and usage.

3.2 SOA Abstract Platform

The SOA abstract platform supports the interaction of various (potentially distributed) service providers through their services. The concept of abstract interaction discussed in [2, 13] is suitable for this purpose. In ISDL, behaviours are defined in terms of (abstract) actions and interaction contributions and constraints on them. Since services only concern observable behaviour, at this level behaviours only contain interaction contributions.

An abstract interaction models the successful completion of a shared activity between the interacting parts, and establishes a result at some location and some time. Constraints can be defined to restrict the results of information established in the interaction, and to restrict which behaviours are allowed to interact with each other. In general, each interacting party constrains the attributes established as result of an interaction: a party may offer a set of values, accept a set of values, or both. These constraints on values supply different ways of cooperation [24], namely, *value passing*, *value checking* and *value generation*. Value passing occurs when an interacting party offers a value and the other parties accept this value. Value checking occurs when all interacting parties offer the same value. In value generation, the interacting parties offer a range of acceptable values and the interaction happens if it is possible to establish a value that matches all requirements. The SOA abstract platform supports only value passing, since this is a more suitable abstraction of the support provided by target platforms.

Fig. 3 illustrates the ISDL notation with a simple service client/provider example. It shows an example of a structured behaviour (of name *Composition*), which consists of five behaviour instantiations (of names *c1*, *c2*, *c3*, *s1* and *s2*) of two behaviour types (of names *ClientBehaviour* and *ProviderBehaviour*). An interaction contribution is represented by a semi-circle drawn on the border of the behaviour in the context of which it is defined.

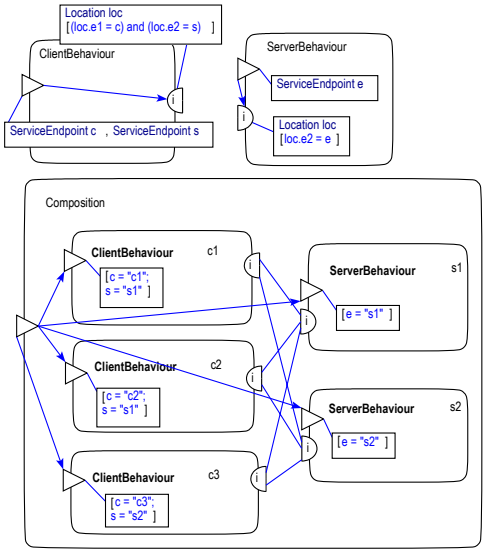


Fig. 3. Example of usage of SOA abstract platform (exported from Grizzle [9]).

In Fig. 3 each interaction is represented by two interaction contributions connected by a line. We use a composite location type (*Location*), which consists of two (interchangeable) service endpoints (*ServiceEndpoint*). A constraint of an interaction contribution is drawn on a box attached to the interaction contribution. In this example, the location constraints are such that servers may interact with any client. The clients constrain location such that *c1* only interacts with *s1*, *c2* only interacts with *s1* and *c3* only interacts with *s2*. Arrows represent enabling causality relations between interaction contributions, and triangles represent entry points that allow behaviours to be instantiated with some parameter values.

Fig. 4 shows the UML class diagram that defines the location attribute type *Location* used at the platform-independent application design level.

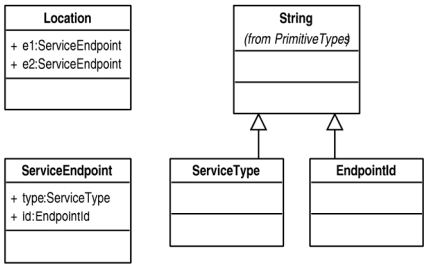


Fig. 4. Location and ServiceEndpoint classes.

3.3 SOA Trader Platform

In order to allow for service discovery, the SOA trader abstract platform contains a trader service, which registers a number of service offers. Fig. 5 depicts the classes relevant to service offers.

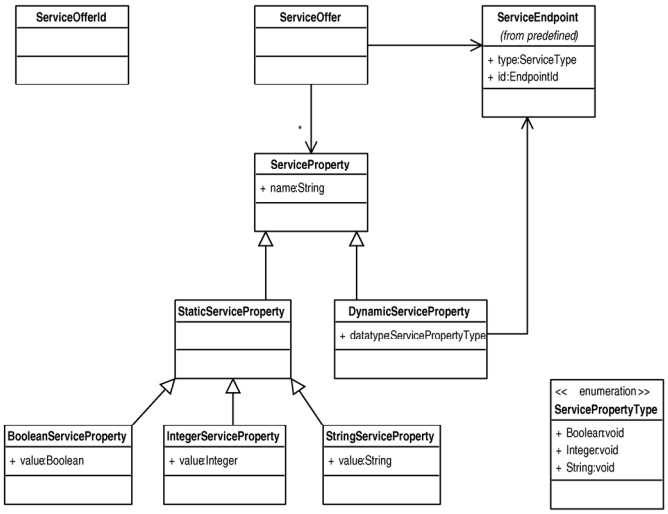


Fig. 5. Service offers.

Service offers (instances of *ServiceOffer*) are represented as information attributes, exchanged with the trader in an *export* interaction. Service offers include a service endpoint (an instance of *ServiceEndpoint*) and a number of service properties (instances of *ServiceProperty*). A service endpoint in a service offer determines how the service represented by this service offer can be accessed. An application part that accesses a service should refer to the service endpoint that corresponds to the desired service. This can be done by properly constraining the location attribute.

Service properties may be either static or dynamic. Static properties have immutable values, which are determined when a service provider exports a service offer. Dynamic properties are evaluated dynamically when a lookup operation is performed [26]. Each static service property consists of a name-value pair. In Fig. 5 these pairs are represented by the attributes of the subclasses of *ServiceProperty*. Each dynamic service property consists of a service endpoint (instance of *ServiceEndpoint*) and a service property type (value of the *datatype* attribute). The service endpoint associated to a dynamic service property is used by the trader to inspect the current value of the dynamic property. The service property type identifies the type of the dynamic property.

A client of the trader service specifies a service query by providing a service type (*ServiceType*) and an expression (*ServiceQueryExpression*) involving service properties (*ServiceProperty*). *ServiceQueryExpression* includes support for basic arithmetic and Boolean operators. The definition of *ServiceQueryExpression* is omitted here due to space restrictions (we refer to [6] for details).

Fig. 6 depicts the behaviour definition of the trader service in ISDL. A *reqServiceQuery* interaction is followed by the execution of the *PropertyEvaluation* behaviour that evaluates the service query expression. Its *exit_offers* exit parameter represents a sequence of offers that comply with the service query.

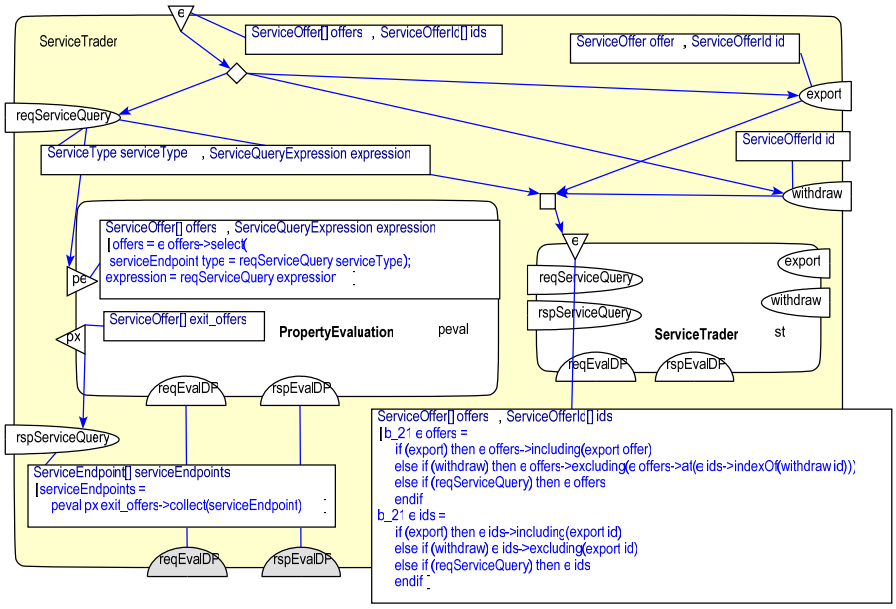


Fig. 6. ServiceTrader behaviour.

The *rspServiceQuery* interaction returns the list of endpoints for the service offers in *exit_offers*. The list of current offers (*offers*) is updated in a recursive instantiation of the *ServiceTrader* behaviour: the occurrence of *export* results in the inclusion of the exported offer (*export.offer*) in *offers* and the occurrence of *withdraw* results in the exclusion of the offer. In Fig. 6, a diamond represents a choice and a square represents a disjunction of enabling relations.

Fig. 7 shows the *PropertyEvaluation* behaviour definition. This behaviour evaluates the service query expression for each service offer and is specified by recursive instantiation. A service offer is only included in *exit_offers* when the service query evaluates to *true* for that particular offer. When the evaluation of a service query requires the evaluation of dynamic service properties, the *DynamicPropertyEvaluation* behaviour is instantiated. Since the recursively instantiated *PropertyEvaluation* behaviour is directly enabled, this recursive instantiation pattern does not force a particular order for service property evaluation: all service properties are evaluated independently, and the results are combined with a conjunction (a filled black square in the ISDL notation).

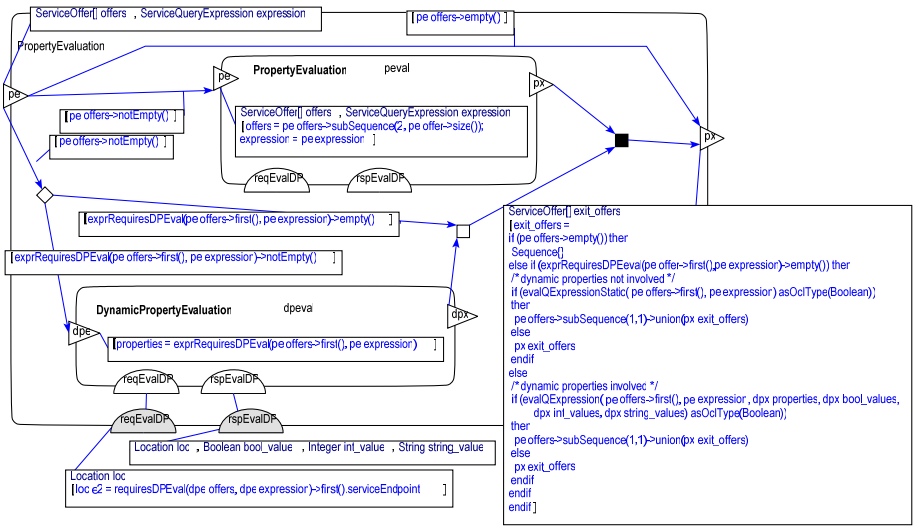


Fig. 7. PropertyEvaluation behaviour.

Fig. 8 shows the *DynamicPropertyEvaluation* behaviour definition. This behaviour is also defined by recursive instantiation, using the same instantiation pattern that was used for *PropertyEvaluation*. For each dynamic property, two interactions occur: *reqEvalDP* and *rspEvalDP*. These interactions occur at the endpoint registered in the service offer as a dynamic property evaluator.

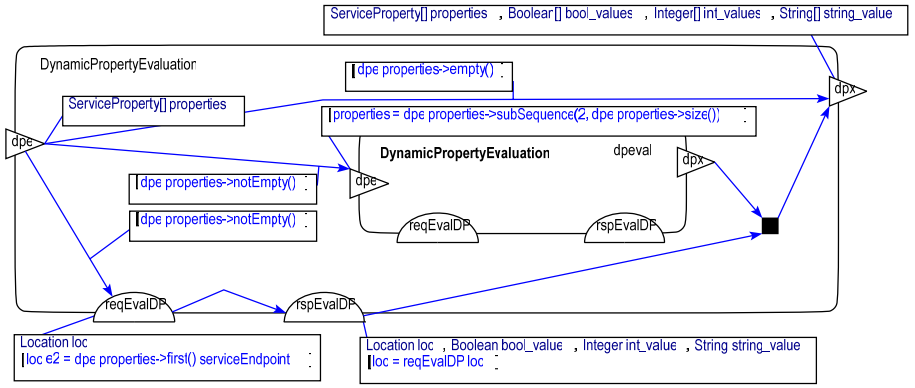


Fig. 8. DynamicPropertyEvaluation behaviour.

The following OCL definitions have been omitted here due to space limitations: *evalQExpression* and *evalQExpressionStatic*, which are used in *PropertyEvaluation* to determine whether an offer complies with a service expression; and *exprRequiresEval*, which is used select properties that must be evaluated in order to evaluate the expression. The complete trader specification can be found in [6]. All constraints in the specification are defined as follows: the left-hand side consists of the name of the (location or information) attribute being constrained; and the right

hand side consists of a side-effect-free OCL expression. The expression determines the value of the constrained attribute. This simplifies significantly the evaluation of constraints in the simulation of the service behaviour.

4 Transformation Patterns

In this section, we discuss the transformation patterns related to the SOA trader abstract platform. As an example application we consider a printer service.

4.1 From Application Service Specification to Platform-Independent Application Design

We assume that an interaction *printReq* is defined at the application service specification level, which determines that some client has requested to print some document. In this example, the client of the printer service defines the maximum size of the queue it is willing to accept. This is done by using a combination of a value passing and value generation interaction (in accordance with the terminology of Section 3.2): the document is passed to the printer service and the size of the queue is determined possibly after consulting the queue length of many different printers, taking into consideration the interaction constraint of the maximum queue size imposed by the printer client. The actual size of the queue determines whether the interaction is successful or not. The use of this kind of interaction is only allowed at the application service specification level. Fig. 9 shows the *PrinterClient* and the *PrinterService* at the service specification level.

At the platform-independent application design level, the original interaction corresponds to a sequence of three (value passing) interactions: a request to the service trader, a response from the service trader and the actual interaction. Expressions on service properties in the query to the service trader are derived from information attributes and their constraints at the service specification level. This derivation requires marking of the service specification to indicate which information attributes should be used in the service query (in this case, the attribute *queueSize*). The interaction occurs at a service endpoint according to the response issued by the service trader. Fig. 9 also shows the *PrinterClient_* and the *PrinterService_* at the platform-independent application design level (the trader service is omitted because of space limitations). The *queueSizeReq* and *queueSizeRsp* are used to evaluate the queue size dynamic property.

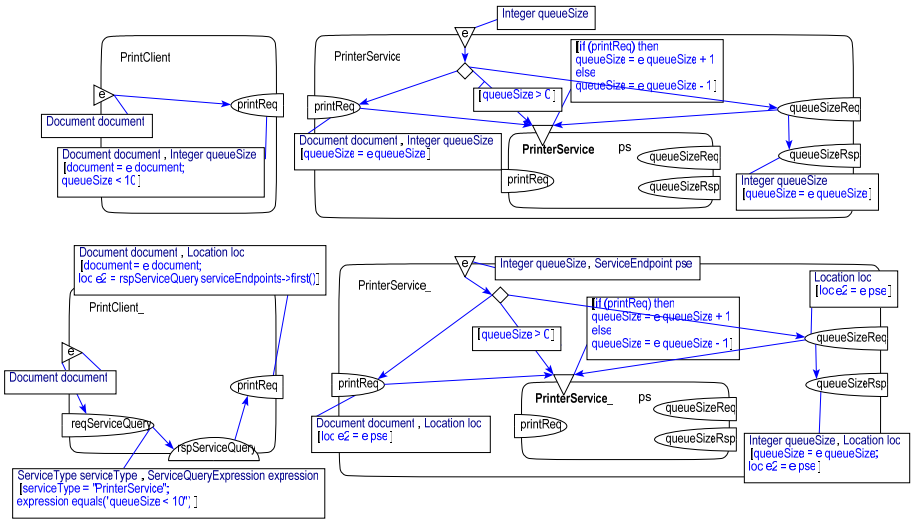


Fig. 9. *DynamicPropertyEvaluation* behaviour.

The decision to implement the abstract *printReq* interaction as combination of a query and the actual print request may not be formally correct according to our refinement rules. This is because we cannot guarantee that the actual queue size at the time of the print request at the lower abstraction level is smaller than the maximum queue size, as prescribed in the most abstract specification. However, this implementation is an acceptable approximation if (i) the time between the *reqServiceQuery* and the *printReq* in behaviour *PrintClient_* is negligible compared with the rate at which jobs are submitted to the printer, and (ii) the SOA trader is capable of timely updating the dynamic properties.

4.2 From Platform-Independent Service to Platform-Specific Service

In order to show the flexibility of the relation between the platform-independent application design level and the platform-specific application design we describe below a possible transformation of platform-independent application designs into two different middleware platforms, namely, Web Services and CORBA. These platforms differ significantly with respect to their support for service discovery.

CORBA provides a trader [17] that supplies a constraint language that allows one to define expressions that correspond to *ServiceQueryExpression* attribute values. In [6], a textual syntax for a *ServiceQueryExpression* has been defined such that any *ServiceQueryExpression* in this form is identical to an expression in the OMG trader constraint language. Furthermore, the OMG trader also supports dynamic service properties. A service exporter must implement the *DynamicPropEval* IDL interface [17]. This interface includes an *evalDP* operation, which receives as a parameter the property name and the required return type. The *evalDP* operation returns the value of the property.

In the case of Web Services technologies, service discovery is provided by UDDI [15]. UDDI does not support dynamic service properties and supports no query language, being able only to provide the values of static service properties (*tModels* [15]) to its clients.

A realisation of the trader service in CORBA is rather straightforward and does not require decomposition of the trader service. A realisation of the trader service in UDDI is more complex due to the differences in the support provided by UDDI and the trader service as specified in the abstract platform. We approach this by introducing a service decomposition step prior to realisation. Fig. 10 shows the two approaches to platform-specific realization. In the case of the CORBA realisation, only one platform-independent application design level is used (level 1 in Fig. 10). In the case of the Web Services/UDDI realization, both platform-independent application design levels 1 and 2 are used.

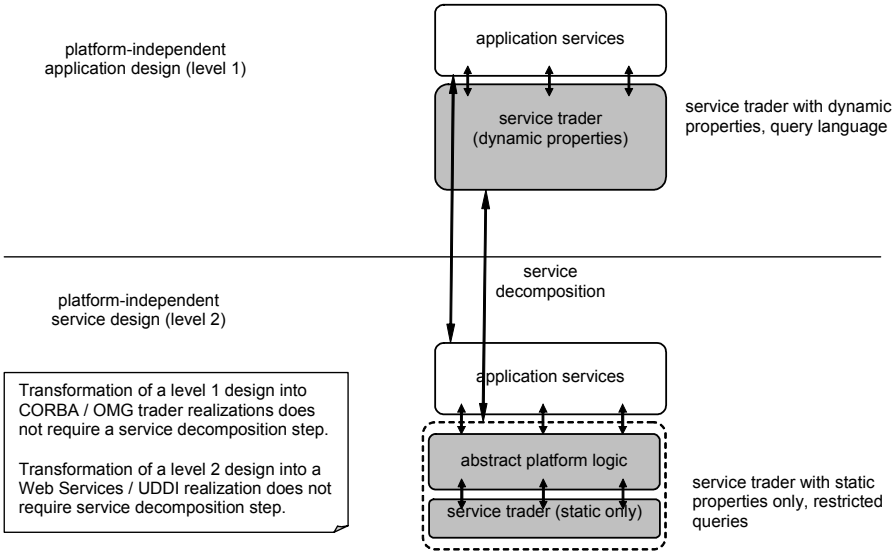


Fig. 10. Realization of the SOA trader platform into two different platforms.

The abstract platform logic must bridge the gap between the trader service at the abstract platform and the service provided by a UDDI registry. Each service offer is registered as an entry in the UDDI registry. Given a query, the abstract platform logic uses the UDDI registry to retrieve all entries for a particular service type, evaluates the expressions (which may include dynamic property evaluation) and returns the list of service offers for which expressions evaluate to *true*. In order to support dynamic service properties, Web service endpoints that are used to evaluate dynamic properties must be registered as an additional *tModel*, which is present only for dynamic service properties.

5 Conclusions and Future Work

We have discussed how services can be modelled in a platform-independent manner, using a combination of a general-purpose behaviour modelling language (ISDL) with UML class diagrams and OCL constraints. The result is an abstract platform for platform-independent application designs based on the SOA principles. We have applied the modelling technique for the trader service, introducing the service discovery pattern at the platform-independent level. The trader service supports dynamic service properties and a simple constraint language for service queries.

We stress that the trader service specification in ISDL defines constraints on the interactions of a client with the trader, without prescribing any internal details of the trader. This is compatible with the service-oriented design principle that services only concern observable behaviour [22]. This gives us maximum flexibility for the further decomposition of the trader, as shown by the realisation of the service trader into a more rudimentary trader (UDDI, featuring no constraint language and no dynamic service properties). This realisation illustrates how target platform differences can be accommodated in the platform-specific realisation step. Further, our specification of the trader service is such that no particular strategy for evaluation of static or dynamic properties is implied. This allows different strategies to be adopted at platform-specific realisation level.

We have used ISDL to model the behavioural aspects of services for four main reasons. Firstly, ISDL supports a broad spectrum of abstraction levels which allows us to cover from service specification to service design seamlessly. Secondly, the concept of abstract interaction in ISDL enables us to capture service designs in a middleware-platform-independent manner (as shown in [2]). Thirdly, ISDL allows to capture causality relations between interactions without constraining the internal implementation of services. And, finally, conformance rules have been defined [23] which can be used to verify whether service designs respect service specifications.

Most approaches to MDA and SOA in literature ignore the description of the behaviour of individual services, specifying individual services solely based on messages exchanged (e.g., described in WSDL or UML class diagrams [10]), or focusing solely on the orchestration of multiple services (e.g., [12]). A consequence of this is that properties of the composition of services cannot be derived from specifications and specifications cannot be simulated. The modelling techniques we have discussed in this paper addressed both aspects.

We have focused on the behavioural aspects of the SOA trader abstract platform and we have not considered the typing system for the trader service. A natural extension of the work reported in this paper is the support for taxonomies and service typing rules.

We have used the Grizzle tool [9] to simulate the trader service specification. Further work on the tool support will involve integrating this tool with support for MOF QVT transformations [20], which will allow us to specify and execute the transformations discussed in this paper in generic model transformation tools. Currently some experiments with a transformation similar to that of section 4.1 have been reported in [6] using GReAT model transformations [1].

Acknowledgements

This work is part of the Freeband A-MUSE project (http://a-muse.freeband.nl). Freeband is sponsored by the Dutch government under contract BSIK 03025.

References

1. A. Agrawal, G. Karsai, A. Ledeczi, "An end-to-end domain-driven software development framework". In: Proc. 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'03). ACM Press (2003) 8–15
2. Almeida, J.P.A., Dijkman, R., Ferreira Pires, L., Quartel, D., van Sinderen, M.: Abstract Interactions and Interaction Refinement in Model-Driven Design. In: Proceedings Ninth IEEE EDOC Conference (EDOC 2005), IEEE Computer Society Press, Sept. (2005) 273–286
3. Almeida, J.P.A., van Sinderen, M., Ferreira Pires, L., Quartel, D.: A systematic approach to platform-independent design based on the service concept. In: Proceedings Seventh IEEE Int'l Conf. on Enterprise Distributed Object Computing (EDOC 2003). IEEE Computer Society Press (2003) 112–123
4. Almeida, J.P.A., Dijkman, R., van Sinderen, M., Ferreira Pires, L.: Platform-Independent Modelling in MDA: Supporting Abstract Platforms, in Proceedings Model-Driven Architecture: Foundations and Applications 2004 (MDAFA 2004), Linköping University, Linköping, Sweden, (2004) 219–233. Revised version appeared in Lecture Notes in Computer Science, vol. 3599, Springer (2005) 174–188
5. Almeida, J.P.A. Dijkman, R. van Sinderen, M., Ferreira Pires, L.: On the Notion of Abstract Platform in MDA Development, In: Proc. 8th IEEE Int'l Conf. on Enterprise Distributed Object Computing (EDOC 2004), IEEE Computer Society Press, Sept. (2004) 253–263
6. Almeida, J.P.A., Iacob, M.E., Jonkers, H., Quartel, D.: Platform-Independent Modelling of Service Infrastructure Components, Freeband A-MUSE/D1.6, TI/RS/2005/078, Telematica Instituut, Enschede, The Netherlands (2005); https://doc.telin.nl/dscgi/ds.py/Get/File-59319
7. Dijkman, R.M.: Consistency in Multi-Viewpoint Architectural Design, Ph.D. thesis, University of Twente, The Netherlands (2006)
8. Erl, T.: Service-oriented architecture: Concepts, technology, and design. Prentice-Hall (2005)
9. Grizzle, http://isdl.ctit.utwente.nl/tools/grizzle
10. Grønmo, R., Skogan, D., Solheim, I., Oldevik, J.: Model-driven Web Services Development. In Proceedings IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-04), Taipei, Taiwan (2004) 42–45
11. Kutvonen, L.: Achieving Interoperability through ODP Trading Function, In: Proc. 2nd Int'l Symposium on Autonomous Decentralized systems (ISADS 1995), IEEE Computer Society Press, Apr. (1995) 63–69
12. Mantell, K.: From UML to BPEL, Model Driven Architecture in a Web services world, IBM (2005) http://www-128.ibm.com/developerworks/webservices/library/ws-uml2bpel/
13. ISDL home, http://isdl.ctit.utwente.nl/
14. ITU-T / ISO: ODP Trading Function: Specification, ITU-T Recommendation X.950 | IS 13235-1 (1997)
15. OASIS: OASIS - Committees - OASIS UDDI Specifications TC; http://oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

16. Object Management Group: MDA-Guide, Version 1.0.1, omg/03-06-01 (2003)
17. Object Management Group: Trading Object Service Specification, V1.0, formal/00-06-27 (2000)
18. Object Management Group: Unified Modelling Language: Object Constraint Language version 2.0, ptc/03-10-04 (2003)
19. Object Management Group: UML 2.0 Superstructure, ptc/03-08-02 (2003)
20. Object Management Group: 2nd revised submission to the MOF 2.0 Q/V/T RFP, ad/05-03-02 (2005)
21. Papazoglou, M.P., Georgakopoulos, D.: Service-oriented computing. In: Communications of the ACM, Vol. 46, No. 10 (2003) 24–28
22. Quartel, D., Dijkman, R., van Sinderen, M.: Methodological support for service-oriented design with ISDL. In: Proceedings of the 2nd International Conference on Service-Oriented Computing (ICSOC) (2004) 1–10
23. Quartel, D.: Action relations Basic design concepts for behaviour modelling and refinement, Ph.D. thesis, University of Twente, Enschede, The Netherlands (1998)
24. Quartel, D., Ferreira Pires, L., van Sinderen, M., Franken, H., Vissers, C.: On the role of basic design concepts in behaviour structuring. In: Computer Networks and ISDN Systems, Vol. 29, No. 4 (1997) 413–436
25. Quartel, D. Ferreira Pires, L., van Sinderen, M.: On Architectural Support for Behaviour Refinement. In: Distributed Systems Design, Journal of Integrated Design and Process Science, Vol. 6, No. 1. Society for Design and Process Science (2002)
26. Vinoski, S.: Service Discovery 101, in IEEE Internet Computing, IEEE Computer Society, Vol. 7, No. 1 (2003) 69–71
27. World Wide Web Consortium: SOAP Version 1.2 Part 1: Messaging Framework, W3C Proposed Recommendation (2003); http://www.w3.org/TR/soap12-part1
28. World Wide Web Consortium: Web Services Description Language (WSDL) 1.1, W3C Note (2001); http://www.w3.org/TR/wsdl

ATC: A Low-Level Model Transformation Language

Antonio Estévez[1], Javier Padrón[1], E. Victor Sánchez[1] and José Luis Roda[2]

[1] Open Canarias, S. L., C/. Elías Ramos González, 4 - Oficina 304,
38001 Santa Cruz de Tenerife, Spain
{aestevez, jpadron, vsanchez}@opencanarias.com
http://www.opencanarias.com
[2] Dpto. Est., I.O. y Computación, ETSII, Grupo Taro, Universidad de La Laguna,
Camino San Francisco de Paula S/N, Campus Anchieta,
38271 La Laguna, Spain
jlroda@ull.es
http://www.taro.ull.es

Abstract. Model Transformations constitute a key component in the evolution of Model Driven Software Development (MDSD). MDSD tools base their full potential on transformation specifications between models. Several languages and tools are already in production, and OMG's MDA is currently undergoing a standardization process of these specifications. In this paper, we present Atomic Transformation Code (ATC), an imperative low-level model transformation language which decouples user transformation languages from the underlying transformation engine. Therefore work invested on this engine is protected against variations on the high-level transformation languages supported. This approach can ease the adoption of QVT and other language initiatives. Also it provides MDA modeling tools with a valuable benefit by supporting the seamless integration of a variety of transformation languages simultaneously.

1 Introduction

In the past few years, Model Driven Software Development, MDSD has successfully positioned as one of the most promising strategies in the future of software engineering [3]. OMG's MDA [2, 7] is a MDSD approach to software development based on models and aimed to provide automation through the various phases in the software development process lifecycle. It follows several standards, which include MOF [9] for describing metamodels, UML [10] for systems modeling and XMI [11], which can describe UML models in the XML format and promote tools' interoperability.

Model transformations in particular allow MDA to achieve automatic model evolution and code generation, and to generally increase productivity in software systems development. Languages to express these transformations have been devised and both commercial and open-source supporting tools capable of performing these activities have been released and are already used in production.

The MOF 2.0 Query, Views and Transformations specification, QVT [8] is particularly relevant to MDA, as it is an attempt to standardize the activities related to

automated transformations between models. This specification will define automatic ways to apply queries, obtain views and execute transformations on models.

Experience gained during the development and handling of our transformation tool's first version, the Business Object Adaptor, BOA [12], has helped us detect several problems that we have solved in our new environment, which is now based on metamodeling and elaborating transformations as opposed to the former procedure of performing transformations in a single, monolythic big step based on XSLT templates.

The long-term goal we pursue is to develop a fully compliant MDA IDE, so the issue of transformation standards affects us deeply. The topic we'll discuss here is the ATC model transformation language. It was born with the aim of shielding our implementation work against changes in the specifications of those supported transformation languages, which inevitably change over time. Now it serves as the low level ground upon which higher-level languages, including the ones that are to become standards, will be integrated in our framework.

This paper is structured as follows: Section 2 briefly outlines the main characteristics of the latest QVT-Merge group standard proposal. Section 3 details ATC and its main components. Section 4 shows an ATC transformation syntax example. Section 5 further discusses ATC issues. In Section 6 related and future work are presented. We end with the conclusions in section 7.

2 QVT-Merge Submission v2.0

In 2002, OMG opened the QVT standardization process with the publication of MOF 2.0 QVT-RFP. As many as 8 proposals were submitted by different organizations and companies during the subsequent two years [6]. In time these organizations gradually converged into one single group, known as *QVT-Merge*. When the third review of this group's proposal (version 2.0) [13] was published, the latest to date, the group was already backed by almost every previous submitter in an effort to speed up the standardization process. At the time of this writing, this third review of the submission of the QVT-Merge group has become the only proposed candidate for the future QVT standard and it is expected that the final QVT standard will quite resemble, or at least be derived from this proposal, so here we show some of its main features.

2.1 QVT-Merge Layers and Languages

The QVT-Merge proposal covers a thorough specification of three transformation languages. The *Relations* and *Core* languages are declarative, and the *Operational Mappings* is imperative.

The specification contains the abstract and concrete syntaxes and the semantics of the three languages. Their detailed metamodels are shown in UML class diagram notation, and concrete syntaxes are described in EBNF. Hybrid collaboration between the imperative and declarative languages is also specified, along with supporting mechanisms for black-box invocations (see figure 1).

The proposal describes steps to convert *Relations* instances into equivalent *Core* syntaxes, which are lower-level. The idea behind this seems to be that compatible tools need only interpret transformation definitions written in *Core*, as those specified in *Relations* could be supported by compiling them to an analogous *Core* syntax.

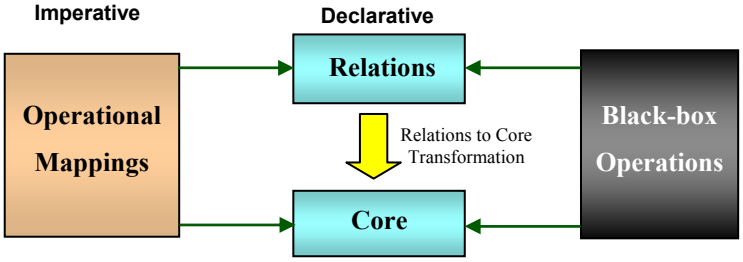


Fig. 1. Description of the QVT-Merge Languages.

3 Introducing the ATC Language

Usually there is no freedom in the election of transformation languages inside tools, since it's hard if not impossible to use any other than the one they come with by default. This also means that while MDA consolidates a QVT standard, tools that opt to give it support are expected to face a rather high amount of refactoring in some cases.

Our primary goal is to provide a tool based as much as possible on industry standards. This is why the pending QVT standard has become a serious problem for us, as the election of supported transformation languages at release time is unclear.

In the meantime we've focused on coding parts that are not affected by the final set of supported languages selected. For instance, the implementation of the lowest-level model transformation mechanisms, and on the establishment of a management infrastructure around models and metamodels. This was the first part of the work and what has emerged from it is ATC and its related transformation engine.

ATC (Atomic Transformation Code) is a general purpose model transformation language designed to operate at the lowest possible level of abstraction, so it is a somewhat harsh, verbose language. Above ATC, those higher-level transformation languages subject to receive support in our environment, will be accommodated through compilation. They are channelled and integrated into our framework to reach the ATC underlying transformation engine, named Virtual Transformation Engine (VTE) by being parsed and compiled into a set of equivalent ATC instances. This is why ATC must be Turing complete in the lax sense.

The engine VTE was created with the sole purpose of understanding and executing ATC instances, which carry information about how to transform models, and it is already being applied successfully in several projects.

There's currently an implementation of ATC and VTE in Java over Eclipse and the Eclipse Modeling Framework (EMF) [4] platforms. Its architecture is depicted in figure 2. Metamodels in this environment are described in terms of the EMF metametamodel, named *Ecore*, and equivalent to MOF.

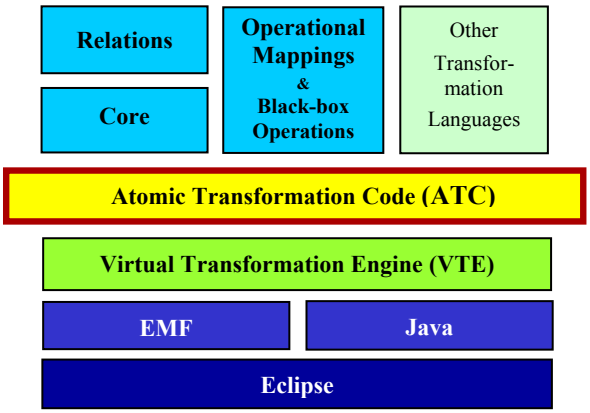


Fig. 2. Architecture layering of a particular transformation environment involving ATC.

3.1 Elements of a Transformation Instance

An ATC transformation consists mainly of:

- an arbitrary number of parameters which represent the participant models,
- identifiers for their related metamodels,
- a set of functional operations that store a hierarchy of semantic objects,
- ATC atoms, the semantic objects in question, each with its own runtime state

Model parameters for a transformation can be read-only, modifiable or created from scratch. Any sort of configuration is supported, such as a single existing model to be modified, or a new one to be created from scratch, one-to-one or many-to-many configurations, and anything that goes in-between.

Execution starts with the transformation's *main* operation call. Like every other ATC functional operation, *main* contains a body filled with atoms arranged sequentially. Some atom types are designed to hold others inside, so finally we get a hierarchy of objects representing the whole ATC transformation information. Therefore it is possible to base the persistence of an ATC transformation as a model serialization, for instance, in an XMI format file, just like EMF does. Interoperability of ATC instances with other tools will be granted as long as XMI compatibility is guaranteed.

Atoms check their state when executed to properly carry out their duties. For instance, state information often tells the engine which model fragments are to be processed. Each atom represents a kind of indivisible byte code with a minimum degree of abstraction, which is why we give it the atomicity condition.

3.2 Language Description

Currently the ATC metamodel contains over one hundred elements among enumerations, data types and classes. As it is impossible to discuss them all here, we'll summarize a classification of the ATC element types.

ATC Expressions. A class identified as an atom type represents a particular atomic transformation semantic unit. Each atom type inherits from a base class named *Expression*, which contains an abstract method with the following signature:

AtcExpression act(TransformationContext tc)

in the reference VTE implementation, which encloses the particular semantic information that makes atom types distinct from each other. It usually comprises no more than twenty lines of code. The *tc* parameter keeps track of contextual information, which includes the local variable registry, the calling stack, as well as parameters and additional runtime information the engine needs to execute the transformation accordingly.

Execution Flow. A list of expressions related with the execution flow include, but is not limited to: *AssignVars*, *Block*, *ExceptionThrow*, *FlowOpReturn, ForEach*, *GetObjectsOfType*, *If*, *InvokeOp*, *InvokeTransformation*, *While*. For instance, a *Block* atom simply encloses a list of atoms to be executed sequentially. *ForEach* takes a data collection as source, usually containing model elements, and traverses it to apply certain actions (which can in turn be an InvokeOp or a *Block* carrying further atoms). *FlowOpReturn* works similar to a return statement but its effect on execution flow is indirect.

Model Transformation. Atom types dealing with model handling and modification include: *CloneModelObject, CreateDataType, CreateModelObject, CreateModel, GetStructuralFeature, SetStructuralFeature*, and those that deal with the contents of lists or other collection types, which are relevant for attributes with multiplicity > 1.

Query and Pattern Matching. Atom types for queries provide us with means to organize data on models and reach particular model types: *GetAllModelElements, GetObjectsOfType*, *GetModelExtent*. The last one delivers the root elements that form a model fragment. Its state information includes the local variable identifier that refers to the piece of model to be queried. Pattern matching is performed explicitly in ATC. It can be achieved if we have means to apply reflectivity over model elements. ATC comes with *IsOfType*, which can be programmed to either perform exact type match or to detect subclasses of a particular model type.

ATC Specific Types. ATC comes with: *Bool*, *Float*, *Int*, *String*. Types in ATC also subclass *Expression*, so their instances are also atoms, and as such, can become the return value of an atom's *act* execution. *String* atoms come enhanced to support common string operations. Among them we find upperization of selective parts, length delivery and substring support. *Null* is a special ATC type that does nothing on its own but can help us detect null assignments in local variables.

Arithmetic and Logical Expressions. Many atom types are built around the ATC specific types. Most of them give support to arithmetics: *Add, Subtract, Multiply, Divide* and *Modulus*, and logics: *And, Or, XOr, Negate, Equals*. These atoms deal transparently with both the ATC types and the primitive types of the native language in which the engine is programmed. Thanks to the *Expression* nature of ATC types and this transparency, these operations can be chained to provide a single final result from a group of combined operations without having to store partial results.

There are still other metamodel elements left, like those that are directly related with the transformation itself: *Transformation, ModelParameter*, *Metamodel*, ...

4 Transformation Examples

We have built a compiler for the *Operational Mappings* language, which is currently quite mature. Certain issues in its EBNF definition and several ambiguities found in its syntax and semantics specification have been sorted out. Future modifications made to the language will be incorporated in the compiler so it adjusts its output accordingly. Obsolete ATC instances will be recompiled. As long as the VTE engine remains unmodified, any other high-level language already supported gets unaffected.

In this section, two small pieces of the *Encapsulation* transformation are presented both in the QVT-Merge Operational Mappings language and its equivalent ATC instances. To make things more interesting, we'll show the ATC transformation definition version from the beginning.

4.1 Mapping Definition

Text in Operational Mappings.

```
mapping inout Property::privatizeAttribute () {
    visibility := "private";
}
```

ATC Equivalent.

```
<atc:AtcTransformation xmi:version="2.0"
   [...] xmlns:atc="http://boa.opencanarias.com/atc/0.5"
name="Encapsulation">

  <metamodels name="UML2">
    <packagesNsURI>http://mset.opencanarias.com/uml2/1.0.0/UML2
    </packagesNsURI>
  </metamodels>

  <modelParameters name="uml2Model" dirKind="inout"
    metamodelId="UML2"/>

  main="//@ownedOperations.7">

  <ownedOperations xsi:type="atc:AtcMapping"
    name="privatizeAttribute">
    <formalParameters xsi:type="atc:AtcMappingParameter"
      name="a" dirKind="inout" typeQualifNm="UML2::Property">
    </formalParameters>
    <mBody xsi:type="atc:AtcBlock">
        <atcAtoms xsi:type="atc:AtcCreateDataType"
          packageNsURI="http://mset.opencanarias.com/
             uml2/1.0.0/UML" dataTypeNm="VisibilityKind"
          sourceString="private" targetId="localVar1"/>
        <atcAtoms xsi:type="atc:AtcSetStructuralFeature"
           ObjectId="a" stFNm="visibility"
           featureVarId="localVar1"/>
    </mBody>
  </ownedOperations>
  <ownedOperations [...]
```

Explanation. The first example consists of the complete definition of a small mapping, where an enumerated type is generated and assigned to a UML2 attribute of a *Property* instance, which represents the contextual parameter for the mapping call.

To keep things clean we have omitted the Operational Mappings' trace information that stores bindings created between model objects during execution, and that can be queried later on during the same transformation. This information is embedded explicitly in the ATC transformation instances during compilation.

Metamodels are defined outside the transformation block. A metamodel is made up of a list of URIs. They refer to Ecore packages in our case. In this example only the URI of our UML 2 metamodel is present. Model parameters for the transformation follow. We can identify the *main* operation as being the operation number 7.

Finally a full list with the transformation functional operations follows. Only *privatizeAttribute* is shown here. Its type is *AtcMapping*. During compilation, the contextual parameter has lost the privileged position it held in Operational Mappings to become an ordinary parameter, the first in the list. The *visibility* assignment, which spans a line in Operational Mappings, has ended up being a *Block* containing two ATC atoms. None of them acts as a container for other atoms.

The first atom obtains a *private* instance of the *VisibilityKind* enumeration type. The second atom will assign it to the *visibility* attribute of the *Property* instance, whose name identifier is '*a*'. The '*localVar1*' variable identifier is used as a *key* in a map of Java variables in order to store its associated value. It is the link established between both atoms. The behaviour of *AtcSetStructuralFeature* is straightforward.

4.2 Mapping Call

Text in Operational Mappings.

```
{
    var attrs := c.ownedAttribute;
    attrs->map privatizeAttribute();
}
```

ATC Equivalent.

```
<atcAtoms
    xsi:type="atc:AtcGetStructuralFeature"
      objectId="c" stFNm="ownedAttribute"
      featureVarId="attrs"/>
  <atcAtoms xsi:type="atc:AtcForEach"
    collectionId="attrs" elementId="localVar12">
    <forBody xsi:type="atc:AtcBlock">
      <atcAtoms xsi:type="atc:AtcInvokeOp"
        op="//@ownedOperations.0">
          <actualParameterIds>localVar12</actualParameterIds>
      </atcAtoms>
   [...]
```

Explanation. The first atom we see here, which is equivalent to the first line in the Operational Mappings sample, defines a new variable, '*attrs*', which represents the list of properties belonging to a given class whose identifier is '*c*'. In the next line a hidden traversal syntax takes place, so that the mapping invocation is produced over every element in *attrs*. This syntax becomes explicit in ATC, as can be seen by the

'*localVar12*' temporary variable managed by the *ForEach* atom. The body for this *ForEach* is merely the *AtcInvokeOp* invocation of the operation number 0, which happens to be *privatizeAttribute*. '*localVar12*' is the actual parameter identifier.

5 ATC Considerations

Imperative Nature. Declarative descriptions are often naturally found in transformation environments. Mappings are established between domain artifacts, and so on. But at the end an algorithmic approach must be followed to reconcile all this information in order to be executed by a machine, so our assumption is that it will be possible to produce ATC explicit imperative representations of those algorithmic semantics. The hard task of evaluating declarative expressions is left to the language compiler. But even if the ATC-based engine is unable to match the execution speed of a direct language supporting engine, we'll have sacrificed performace in favor of a flexible design

Multi-Language Tools. The capability for simultaneous support of different transformation languages in the same tool is very interesting, provided it doesn't bloat its responsiveness and general performance. As there is no such language capable of solving transformation problems in all kinds of situatios with complete flexibility, power and ease of use, the layering principle allows focusing on the integration of a wide range of high level languages at the hands of the architect in the same development environment. Future adoption of new emerging languages will also be possible.

This diversity opens up the possibility for the joint collaboration of several transformation languages in the sense of Domain Specific Languages (DSL) [5]. Reuse of legacy transformation languages, which in turn can see their longevity increased, is also interesting. Entire repositories of transformation definitions can be translated into ATC versions and related metamodels created to assist them. This has to do with the added value the framework earns each time new EMF metamodels are provided and ATC transformations and transformation language compilers produced.

ATC Unfolding to Java Files. Previously ATC execution performance was penalized because of transformations being model objects to be traversed and the *act* method having to be invoked for each executed atomic unit. Other issues, such as ATC's specific types operation handling, like addition, primitive type assignment, and several ones like variable declaration, comparison or execution flow control checks, came into play to further introduce overhead and slow things down.

Recently we have started a new version for VTE. To keep efficiency at maximum, one last compilation step is now performed, this time to unfold ATC XMI instances into native plain Java code. Each functional operation is analyzed and the code associated to each of its composing atoms' types already available in the previous engine version is sequentially dumped inside a method that represents the entire original operation. Specific arrangements are made for each different processed atom type, and loose pieces of code are glued together inside the overall method. The surrounding class becomes the transformation itself. References to metamodels are automatically embedded in the code.

The outcome of this plain java class matches the structural aspect of the original transformation definition, That is, as many methods are defined in the class as functional operations are present in the original transformation instance. The transformation context is treated slightly different now. To spread useful runtime information it just becomes an explicit additional parameter in every operation (including *main*).

This new mechanism not only considerably helps in boosting performance, but now it's easier to promote blind transformation and operation invocations (black-box), once a contract has been established concerning how to access the entrance point of the transformation and to choose models as actual parameters.

Support for QVT Requirements. We won't discuss here issues about supporting QVT recommended features such as traceability, bidirectionality and incrementality in our framework. These are still open areas for investigation in our case, but we expect the ATC language to be agnostic of these features, as support can be expressed explicitly in its transformation instances (similar to the trace classes infrastructure for *Operational Mappings*) or integrated in the environment through the VTE engine or in parallel with it. This helps keep the language structure focused and compact.

6 Related and Future Work

Following the same low-level principle discussed for ATC, another transformation language, ATL [1], has recently added an imperative virtual machine to its layering architecture with a similar abstraction level but different treatment of core types. Similar examples include the QVT-Merge language pair Relations vs Core. Concerning transformation representations as model objects, the QVT-Partners [14] group has an open transformation implementation based on the composition of semantical units.

As future work, it will be interesting to see how ATC is able to deal with the Relations and Core languages. A compiler for Core will soon start development. We also expect support for other languages to be developed through other research groups.

For the moment, we don't plan to port ATC and VTE to other underlying technologies aside from EMF and Java. We expect to be able to apply the MDA paradigm, so when the ATC metamodel matures to include semantics (it is currently hardwired in the engine), versions for other underlying technologies (such as MDR, or C++) will automatically be generated. Other future challenges include exploring migration issues about the integration of ATC and VTE implementations in foreign MDA tools.

7 Conclusions

In this paper we have detailed the ATC model transformation language, its composition and set of instructions, and its underlying execution mechanisms. ATC is a low-level, imperative language designed for model transformations and thus, not quite user-friendly. We've also introduced VTE, its supporting transformation engine,

which doesn't understand any other language and to which a plain java compilation process has recently been added to enhance runtime performance.

We've discussed the role of ATC in the transformation tools as an intermediate layer that assists in the integration of the common abstract transformation languages. Integration of each language is achieved by means of a compilation module that produces semantically equivalent instances in the ATC syntax.

We consider the current QVT-Merge standard proposal a reference regarding high-level transformation languages suitable for common use by transformation engineers. Here its architecture has been depicted. A compiler for ATC specifically tailored to translate *Operational Mappings* instances is already available. Two examples show what ATC instances look like and how they compare to their equivalent high-level language original syntax. We've also presented several other transformation languages each sharing certain similarities with ATC or with the transformation engine design.

We've described how the layered arrangement brings benefits to the unification of a transformation tools' architecture, with a single central engine able to give support to many simultaneous languages. Newer and older languages can coexist in the same environment and even complement each other. Finally this layering can help tools in their adoption of the upcoming QVT standard.

Acknowledgements

This paper has been supported by the *Ministerio de Educación y Ciencia* (PTQ2004-1495) and the *Fondo Social Europeo*. We would like to thank *IZFE (Diputación de Guipúzcoa)*, the *Excmo. Cabildo Insular de Tenerife* and the *DG de Universidades e Investigación del Gobierno de Canarias*, for their overall support of this work.

References

1. ATL, The Atlas Transformation Language, http://www.sciences.univ-nantes.fr/lina/atl/
2. Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture, Practice and Promise. Addison-Wesley (2003)
3. Butler Group Application Development Strategies Report, http://www.butlergroup.com/reports/ads/
4. Eclipse Modeling Framework (EMF), http://www.eclipse.org/emf
5. Fowler, M.: MF Bliki: DomainSpecificLanguage, http://www.martinfowler.com/bliki/DomainSpecificLanguage.html
6. Gardner, T., Griffin, C., Hauser, R., Koehler, J.: A Review of OMG MOF 2.0 QVT Submissions and Recommendations Towards the Final Standard. 1st International Workshop on Metamodeling for MDA, York, UK (2003)
7. Mellor, S., Scott, K., Uhl, A., Weise, D.: MDA Distilled. Principles of Model Driven Architecture. Addison Wesley, 2004
8. OMG, MOF 2.0 Query/Views/Transformations RFP, OMG Document ad/2002-04-10 (2002)
9. OMG, Meta-Object Facility (MOF), http://www.omg.org/mof

10. OMG, Unified Modeling Language (UML), http://www.uml.org
11. OMG, XML Metadata Interchange (XMI), http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI
12. Padrón, J., Estévez, A., Roda, J.L., García, F.: An MDA-Based Framework to Achieve High Productivity in Software Development. Software Engineering and Applications, Track 436-218 (2004)
13. QVT-Merge Group, Revised Submission for MOF 2.0 Q/V/T RFP, OMG Document ad/2005-03-02
14. QVT-Partners, http://qvtp.org

Applications

Model-Driven ERP Implementation

Philippe Dugerdil, Gil Gaillard

Information Systems Department, Haute école de gestion,
University of Aplied Sciences, 7 rte de Drize, CH-1227 Geneva, Switzerland
philippe.dugerdil@hesge.ch

Abstract. Enterprise Resource Planning (ERP) implementations are very complex. To obtain a fair level of understanding of the system, it is then necessary to model the supported business processes. However, the problem is the accuracy of the mapping between this model and the actual technical implementation. A solution is to make use of the OMG's Model-Driven Architecture (MDA) framework. In fact, this framework lets the developer model his system at a high abstraction level and allows the MDA tool to generate the implementation details. This paper presents our results in applying the MDA framework to ERP implementation based on a high level model of the business processes. Then, we show how our prototype is structured and implemented in the IBM/Rational® XDE® environment

1 Introduction

Due to the increasing pressure on IT cost and standardization, a growing number of companies have turned to Enterprise Resource Planning (ERP) systems to build their core IT system. Formerly limited to big companies, the phenomenon also reached the Small and Medium Industries (SMI) [21]. It is now widely accepted that ERP systems provide a viable alternative to custom application development for the standard information management needs and that it is often superior in terms of quality of the implemented business process [8]. If ERP systems have become a true alternative to custom-made IT systems, managers are concerned about the excessive dependency it may leads to the ERP vendor. Seen from the outside, an ERP implementation is overwhelmingly complex. Also, it is seldom the case that the IT team of a customer company masters all the details of its ERP system. In fact, when some large modification must be made to the system, it often requires the help from the ERP vendor's consultants. This may lead the company management to feel that it is "loosing control" over its IT system. A solution is to provide the management with a model of the system which would fit its level of understanding and expertise. In fact, the right modeling level rests at the business process level and should be expressed in some graphical way. However the modeling tool and graphical language should not bring yet another level of dependency. This is why our tool and approach use UML and its extension mechanism to design the standard business modeling elements and semantics. In fact, an UML extension has already been proposed by Eriksson & Penker [6] that closely matches one of the widely used business modeling standard [4]. It complements the

UML Business Modeling Profile published by IBM [13]. We then used the Eriksson-Penker profile and implemented it in a widely available and extensible modeling tool, XDE® from IBM/Rational®. However, for this approach to be successful, one must make sure that the model is aligned with what's implemented. A very promising way to do it would be to generate the parameterization elements from the model itself. Starting from this idea we then investigated the use of the OMG's MDA framework [15,5] to build a semi-automatic ERP customization tool. Starting from a business process model, the tool then gradually refines the model into increasing level of detail down to the database elements necessary to implement the processes on the target system.

2 Related Work

Acknowledging the fact that the mismatch between the enterprises needs and the system customization is one of the reasons for ERP implementation failure [22] and that the proprietary customization language is often hard to manage, some attempts have been made to replace it by some standard graphical language. On the other hand, the need to develop business process models for ERP implementation projects is well known [7]. The Use-Case and Object Oriented (OO) approach to model business processes has been advocated by Jacobson since the mid nineties [12]. Later, the use of UML as a business process modeling language has been widely documented by IBM/Rational consultants [16,3,10]. But the application of the OO concepts to ERP implementation has not been proposed until recently. For example Arinze and Anandarajan [2] proposed an OO framework to ease the customization of an ERP system. However, their approach does not really improve the level of modeling. It mainly replaces the proprietary customization language by some OO representation. But the business processes themselves are not the target of their modeling tool. The modeling of the business processes in a ERP-independent format has been proposed by Sheer [18] who developed the commercial product ARIS [11]. However this approach yet reintroduces a level of dependency, not to the ERP system but to the tool vendor. The use of the OMG's MDA framework to model the enterprise and its processes has been deeply investigated by Wegman [23]. But his model has not been applied to the development or customization of IT systems. Linvald and Østerbye recently proposed the use of UML to implement an ERP system [14]. However, it concentrates on the visual aspects of UML and does not propose any methodology nor does it use the MDA framework. On the other hand, Rolland and Prakash proposed to use UML to model the functional requirement of an enterprise IT system and to compare it to some target ERP system [17]. But this work stays at the specification level and does not deal with the customization problems. Finally, it is worth noting that the advantage of using one unique modeling language, namely UML, for business as well as system modeling has been advocated by Heberling et al. [9]. But their work does not mention the use of UML for ERP implementation.

3 MDA and ERP Implementation

One of the motivations behind the design of the OMG's MDA framework is to promote platform independence when designing IT applications. In fact, the developer will concentrate on the platform-independent features of his application and will let the programming environment generate the details and programming elements according to the chosen target platform. The highest conceptual model, the CIM (Computation Independent Model), is targeted at domain practitioners [15]. It is sometimes called the domain model and includes the main concepts and entities of the domain. Then, by adding the knowledge of the common business processes implemented in any ERP system one gets the Platform Independent Model (PIM). Although the target ERP platform should not be considered at this early stage of the modeling process, one nevertheless knows that the target is an ERP system and not a custom-developed application. This is consistent with the observation of Almeida et al. [1] that the design of the PIM should know the "general capabilities of the potential target platform". In this sense, the MDA framework resembles the best practices in ERP implementation: first design the business process to be implemented then proceed with parameterization [19]. All our models are based on UML and its lightweight extension mechanism: the UML Profile. As UML is becoming standard knowledge for IT engineers, using our technique will save them the burden to learn some proprietary ERP implementation language. Moreover, our approach can be implemented in any commercially available tool which supports the MDA framework, such as IBM/Rational® XDE®.

MDA was initially intended to generate custom made applications. In this case the last step of the process, the transformation from the PIM to the Platform Specific Model (PSM), represents the code generation for the target platform. In the case of an ERP, the system is already implemented. It must only be configured according to the target business processes. This amounts usually to the generation of the parameters in the ERP's tables. Grossly speaking, an ERP system is like a toolbox of components (visual and non visual) to enable/disable and tune according to the process to be implemented. This is why the customization of an ERP differs from code generation: we do not generate or remove components; we only enable/disable components and generate constraints information for the ERP parameterization engine to configure the system. Of course, many ERP customizations include the programming of some specific function using the ERP's integrated programming environment. This could to a large extent be modeled as Object Constraint Language (OCL) expressions. But this very capability is not covered in the present paper as we only target the generation of the ERP parameters.

4 Steps of the Implementation Method

To extend the UML language to include the business process modeling we designed a UML profile that includes some new stereotypes and tagged values [20]. Tagged values are used to propagate the customization information among the MDA models by the MDA transformations. For example, some of the tagged value will tell the

system if a given entity will be enabled or disabled on the target system leading to its presence / absence on the screens.

4.1 CIM and PIM Model Elements

In the case of an ERP, the CIM and PIM models are pre-built and represent the entities of the domain model. In fact, the PIM is not *generated* from the CIM but it is *tuned* according to the target process to implement. The elements of theses models are business entities represented by the *Resource* stereotype, which is an extension of the "Class" metaclass in the UML meta model. According to [6] a resource is an entity which is either consumed or transformed by a process. Moreover, a Boolean tagged value is associated to each of these entities to represent the state of the entity: 'used' or 'unused'. Examples of such resources are: item, order, currency, invoice or loan. Figure 1 shows a subpart of the CIM model built in our prototype system.

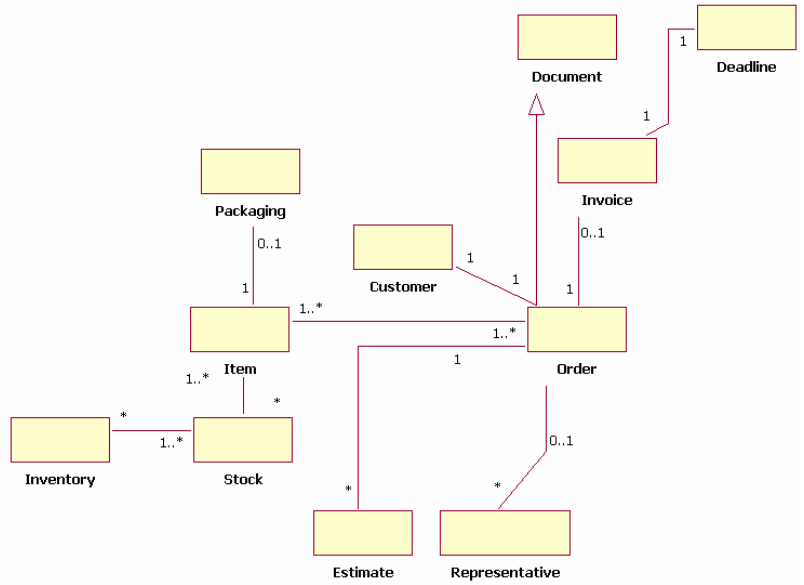


Fig. 1. The CIM model.

4.2 PSM Model Elements

The elements of the PSM are specific to each target ERP system. They are implemented as table rows, code units, forms, screens and reports (whose parameters are usually represented as values in table rows). In our prototype, we targeted Adonix®, one of the leader ERP system in the SMI business segment. Then, the PSM takes the form of value of rows in the Adonix tables.

5 Transforming the CIM to the PIM

On of the key feature of our approach is that the transformation rules from CIM to PIM and from PIM to PSM are themselves represented as models. In fact, our goal with this approach was to make it as easy as possible for the ERP customers to use our tool to configure their system. Then, the CIM to PIM transformation is represented by the high level model of the generic business process to be implemented in the ERP. This model is used to propagate tagged values from the CIM to the PIM depending on the state of each of the business process' tasks. When an engineer must configure a transformation, he will simply select the generic process to be implemented from a library, then select the tasks that must be enabled (used) or disabled (unused) in this process. For example, if a process is disabled, then the linked resources will also be disabled. This will then further propagate to all the processes that use or manipulate these resources. Next, if a resource represents the output of a disabled process, then it is also disabled and all processes that use this resource as input will also be disabled. At the end of this propagation step, any OCL constraints that represent specific limitations, initial values, message or time constraints in the CIM must be copied to the PIM. Figure 2 shows a business process model that represents the CIM to PIM transformation rule in our prototype system, with its associated resources, people, information and goals.

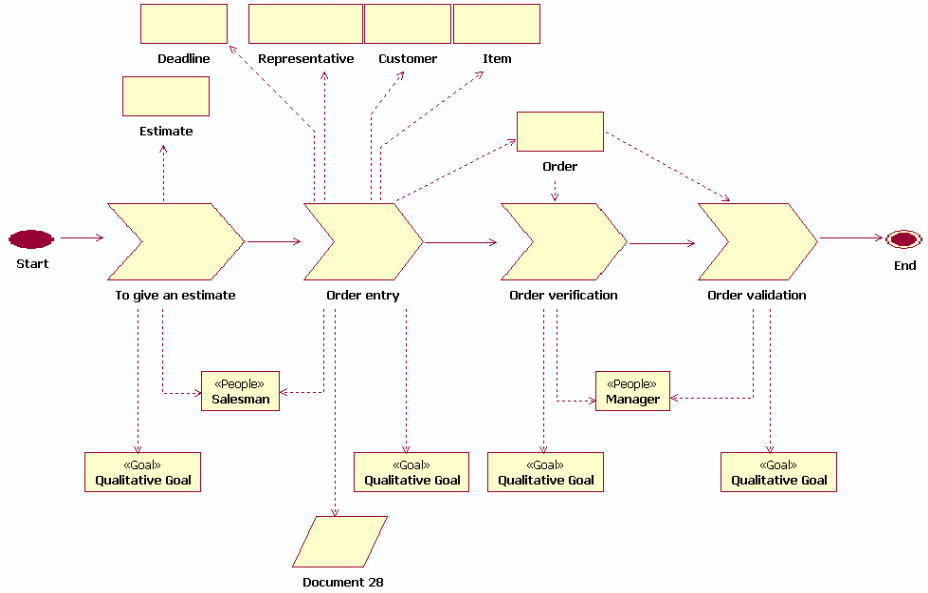


Fig. 2. Transformation rule: a process model.

The business model that represents the transformation rule from CIM to PIM is built from the following elements.

Business process (stereotype): represents a set of activities to be performed by people. It may be structured as a hierarchy of sub processes where the lowest level is the *activity*. The business process is therefore an extension of the "Activity" metaclass (fig 3). However, the activities associated to a given process are platform dependent. Therefore they will be defined when dealing with the PIM to PSM transformation. When building a process model, the resources must be linked to the business process which manipulates them.

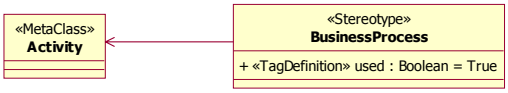


Fig. 3. The Business Process metaclass.

People (stereotype): represents a "human resource" that is associated to a process. It is an extension of the "Class" metaclass (fig 4). This element is used to define authorizations profiles over the system (access rights to the ERP functions). *Goal* and *information* (stereotypes): represent the business goal of a process and the information required to perform a process. These elements must be linked to their business process. For the moment, these two elements are used for documentation purpose only. They both are extensions of the "Class" metaclass (fig 4).

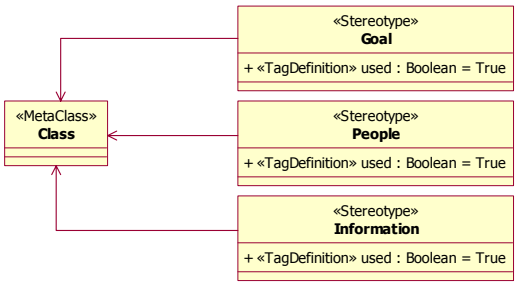


Fig. 4. The metaclasses for resources.

6 Transforming the PIM to the PSM

The PIM to PSM transformation is also represented by a model. In this case it is the model of the actual implementation, in the target ERP system, of the generic business process used as the CIM to PIM transformation. This implementation is represented as a set of *Business Activity* diagrams, each diagram corresponding to one of the tasks of the business process. These models are used to configure the PSM model elements according to the tagged values of the PIM model elements and the status of each of the activities of the activity diagrams of the tasks. In fact, when an engineer must configure a PIM to PSM transformation, he will simply select the status of the activities in the activity diagram that represents the business task to configure. The value of

the status of an activity is dependent on the target ERP system. For example, one may have : unused, optional, obligatory, shown,… In figure 5 we represent a business activity diagram that shows the entity linked to each of the activities. The figure also shows a pop up menu that let the configuration engineer choose the status of one of the activities.

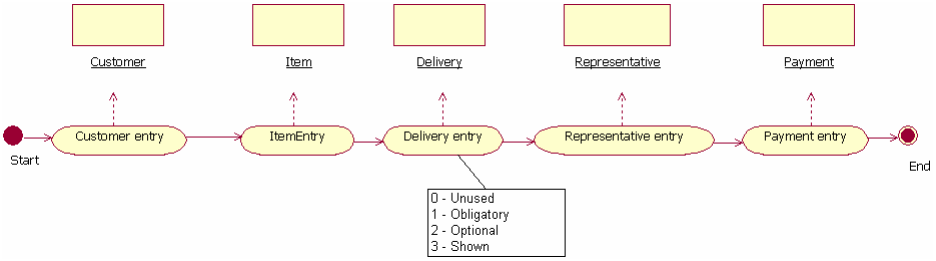


Fig. 5. Business activity diagram for a business process.

The transformation process will propagate the values of the activities' status to all the attributes of the linked business entities, unless the latter were already disabled by the first transformation. For example, if the status of the "Payment entry" activity is set to "Obligatory" then all the attributes of the linked business entity "Payment" entity will be set to "mandatory" through the addition of a new property. From the final status of these entities, the system will generate the parameters for the target platform. Finally any OCL constraints will be processed to generate the constraint in the appropriate target format. For example if, as in Adonix, the target of the customization process is a set of tables, a set of generic SQL scripts will be executed to populate these tables using a mapping file that holds the mapping from the entities and their attributes to the tables' records.

The activity diagram that represents the transformation rule from the PIM to the PSM is built from the *BusinessActivity* stereotype which is an extension of the "Activity" metaclass (fig 6). A business activity may be of type data entry, data validation or data selection. It is linked to the resources it manipulates. A tagged value "status" is associated to each business activity to represent the user-selected status of the activity (see figure 5).

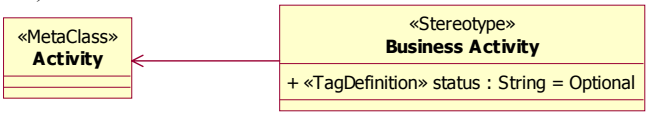


Fig. 6. Business Activity metaclass.

The business activity diagram associated to each business task is ERP dependent. Then, it must be created for each new target ERP platform. Moreover, if the target of the customization process is a set of tables, one must also create the mapping file from the business entities to the tables of the target platform. In figure 7, we summarize the steps of the transformation from CIM to PIM to PSM using the technique we described.

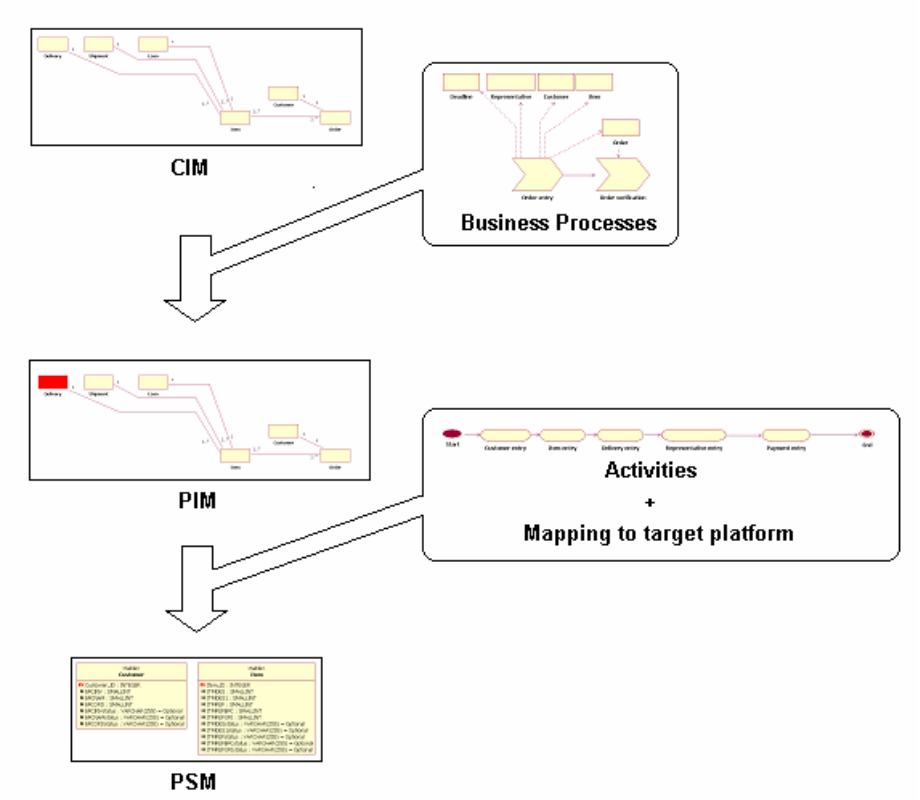


Fig. 7. Summary of the model-based MDA transformations.

7 Implementation of the Prototype

The prototype of our system has been implemented in the IBM's XDE environment augmented with the MDA toolkit (fig 8). Our own extension is built as an eclipse plugin written in Java. When a task of a generic business process is disabled by the configuration engineer or if an entity gets disabled by status propagation, it is turned to another color (red) in the diagram. Then it is easy for the configuration engineer to see the current status of the customization (fig. 9).

8 Conclusion and Future Work

The goal of this project was to validate the applicability of the MDA framework to the customization of an ERP. The use of an ERP system as the target platform of the MDA approach brings some unique constraints. First, the final application is not generated because it already exists. Rather, it must be configured or customized.

Second, the process that could be implemented are dependent on what is available on the target platform. In other words, the spectrum of the possible business process to implement is limited. Third, although the customization language is specific to each ERP, the business process to be implemented can be represented by some standard graphical notation.

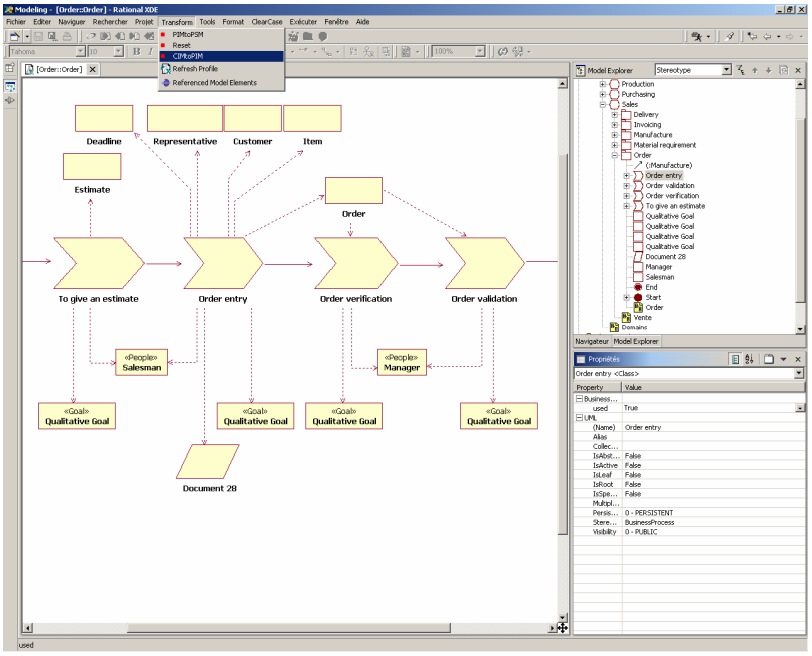


Fig. 8. Prototype in the IBM's XDE environment.

Then we investigated the possibility to use this graphical notation to generate the customization constraints. This lead us to define the MDA transformations themselves as models (business process and business activity diagrams) using a standard notation (BPML, which has been implemented as a UML Profile). This has the unique advantage to free the user from knowing the peculiarities of some specific customization language or system. Using this technique, the customization is self documenting. The impact of any subsequent change could then be easily analyzed at a conceptual (business) level. Although our first prototype only covers a small subset of the business processes of Adonix, we have been able to generate the parameters down to the Adonix tables successfully, only using our graphical notation. These parameters triggered the correct behavior on the system. The next step in this research will be to extend the prototype to the other processes and to further validate the approach by targeting other ERP platforms (Microsoft's Navision®). A new topic of research would be to go the other way around: to generate the high level (business) view from a given ERP implementation. Although one could already foresee the many difficulties of this endeavor, the present research has shown some path toward this goal.

Acknowledgement: this work has been supported by the HES-SO 12493 grant (IS-Net89) from the Swiss Confederation.

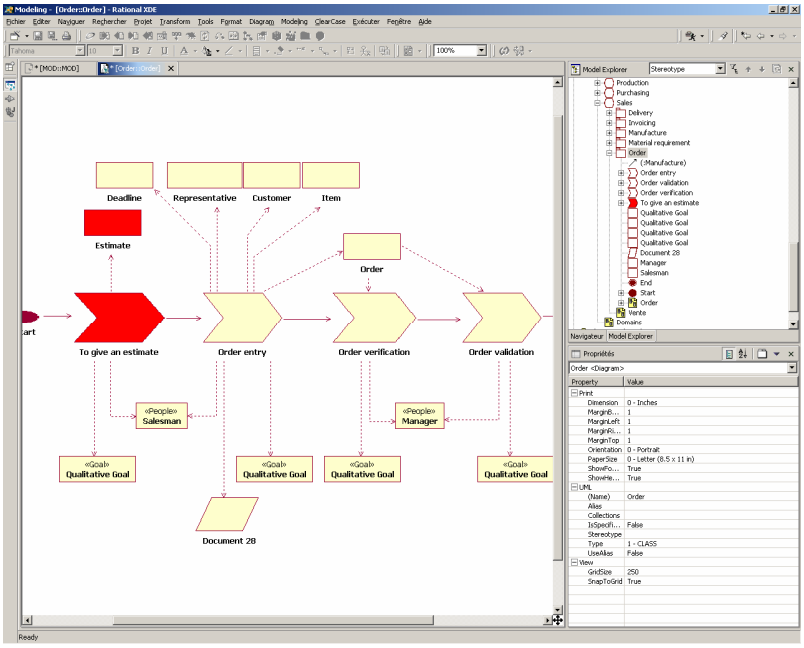


Fig. 9. Business process with disabled elements.

References

1. Almeida J.P., Dijkman R., van Sinderen M., Fereira Pires L.:On the Notion of Abstract Platform in MDA Development. Proc IEEE EDOC (2004)
2. Arinze B. and Anandarajan M.: A Framework for Using OO Mapping Methods to Rapidly Configure ERP Systems. Communications of the ACM Vol. 46(2) (2003)
3. Baker B.: Business Modelling with UML: The Light at the End of the Tunnel. The Rational Edge, Rational Software, December (2001)
4. BPMI.org,: Business Process Modeling Notation - Working Draft 1.0 www.bpmi.org. (2003)
5. Frankel D.S.: Model Driven Architecture. OMG Press. Wiley Publishing, (2003)
6. Eriksson H.-E., Penker M.: Business Modeling with UML. John Wiley & Sons, (2000)
7. Gulla J.A., Brasethvik T.:On the Challenges of Business Modeling In Large Scale Reengineering Projects. 4th International Conference on Requirements Engineering (2000)
8. Harwick T.: Three Half-Truths About Custom Applications, Forrester Inc., November 27 (2002)
9. Heberling M., Maier Ch., Tensi T.: Visual Modeling and Managing the Software Architecture Landscape in a large Enterprise by an Extension of the UML. Second Workshop on Domain Specific Visual Languages, OOPSLA (2002)
10. Heumann J.: Introduction to Business Modeling Using the Unified Modeling Language. The Rational Edge, Rational Software, March (2001)

11. IDS Sheer : From Process Models to Application, ARIS P2A. White Paper. IDS Sheer AG, (2003)
12. Jacobson I., Ericsson M., Jacobson A.: The Object Advantage. Business Process Reengineering with Object Technology. Addison-Wesley (1995)
13. Johnston S.: Rational UML Profile for business modeling. IBM Developerworks. www-128.ibm.com/developerworks/ rational/ library/5167.html (2004)
14. Linvald J., Østerbye K.: UML tailored to an ERP framework. Second Workshop on Domain Specific Visual Languages, OOPSLA (2002)
15. Miller J., Mukerji J.: MDA Guide Version 1.0. omg/2003-06-0. OMG, June (2003).
16. Ng P.-W.: Effective Business Modeling with UML: Describing Business Use Cases and Realizations. The Rational Edge, Rational Software, November (2002)
17. Rolland C., Prakash N.: Matching ERP System Functionality To Customer Requirements. Proc. Fifth International Symposium on Requirement Engineering, (2001)
18. Scheer A.-W., Habermann F.: Making ERP a Success. Communications of the ACM, Vol 43, N°4 (2000)
19. Thomas J.L.: ERP et progiciels de gestion integrés (ERP and Packaged Business Software). Dunod, Paris (2002)
20. UML Unified Modeling Language Specification, Version 1.5, OMG, March (2003).
21. van Everdingen Y., van Hillegersberg J., Waarts E.: ERP Adoption by European Midsize Companies. Communication of the ACM, Vol 43, N°4 (2000)
22. Vogt Ch.: Intractable ERP. A comprehensive Analysis of Failed Enterprise Resource Planning Projects. ACM SIGSOFT, Software Engineering Notes, 27(2), March (2002)
23. Wegman A., Preiss O.: MDA in Enterprise Architecture? The Living System Theory to the Rescue. Proc. IEEE EDOC Conference (2003)

MDA Approach for the Development of Embeddable Applications on Communicating Devices

Eyob Alemu[1], Dawit Bekele[2], Jean-Philippe Babau[3]

[1]MicroLink Information Technology College, P.O.Box 5/1030, Addis Ababa, Ethiopia
eyob_alemu@yahoo.com
[2]Department of Computer Science, Addis Ababa University,
P.O.Box 3479, Addis Ababa, Ethiopia
Dawit@math.aau.edu.et
[3]CITI Laboratory INSA, LION, France. 20, avenue Albert Einstein, 69621 Villeurbanne cedex
jean-philippe.babau@insa-lyon.fr

Abstract. Focusing on the communications subsystem of embedded platforms, this paper introduces an MDA based approach for the development of embeddable communicable applications. A QoS aware and resource oriented approach, which exhibits the runtime interaction between applications and platforms, is proposed. Reservation based (typically connection oriented) networks are specifically considered.

1 Introduction

Recent technological advances are making possible the embedding of both processing and communication functions in highly integrated, low-cost devices such as PDA's and mobile phones. This is promoting the use of a distributed approach in many application fields including embedded systems, which is now leading to the current and future realm of pervasive computing [1]. As communication is extensively used as an interaction medium for such devices, it makes up the most important platform service in such distributed systems. Today, a large variety of networks are currently available to build distributed embedded systems. Moreover, most of them are competing in the same domain of application. For example, CAN [10] and I2C [11] are used in automotive and industrial systems, whereas Bluetooth [8] and IrDA [9] are used for interconnecting peripherals and portable devices. The middleware platforms considered so far in the MDA such as CORBA are heavyweight and do not generally fit the domain of embedded systems. Moreover, resource limitation is a typical characteristic of this domain, which makes the issue of Quality of Service (QoS) a major concern. In this paper, we propose a QoS aware MDA approach for the development of embeddable communicable applications focusing on the communication subsystem. The approach shows an adaptation of the enterprise MDA towards addressing platform variability in the development of applications for embedded devices.

2 MDA and Embedded Systems

With the general MDA specification, systems are first modeled using Platform Independent Models (PIMs). The next step transforms the PIMs to Platform Specific Models (PSMs) through a systematic transformation process. In recent years, the capability of the hardware devices is enhanced to provide extensive interfaces and the possibility of hosting applications of different types. Programmable interfaces and software abstraction layers are becoming possible to support flexible system developments [5]. This evolutionary enhancement of embedded systems from their specific purpose functionality to a more general, multipurpose and more intelligent capability is making the devices not only capable of hosting embedded applications but also communicate with each other to share resources and to transfer information. The current and future vision of pervasive computing can benefit from this advancement since it makes extensive use of embedded devices. Besides the software development complexity of this domain, platform variation is a very critical problem. Moreover, resource limitation has made the development to focus on QoS and platform level issues.

2.1 The Embedded System Platforms

In [4], a definition for an embedded platform is presented as a "family of Micro-Architectures possibly oriented towards a particular class of problems". A recent initiative in this domain is the platform-based approach proposed in [5] and further improved in [4]. Using Platform Based Design approach, the platforms for embedded systems are modeled at different abstraction levels so that developers could choose the appropriate abstraction level that can avoid their concern about the details of the platforms. A typical layered architecture of an embedded platform is shown below (Fig 1) [4].

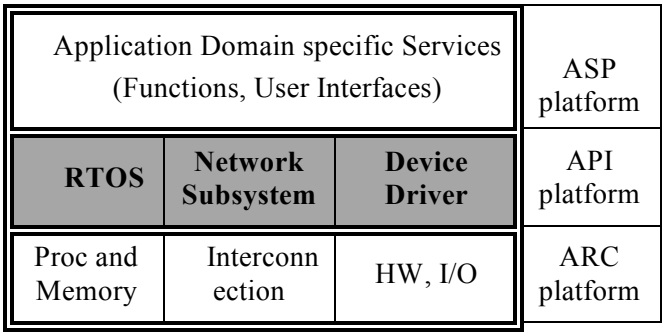


Fig. 1. Platform descriptions at different levels.

As shown in Fig 1, the ARC Layer includes a specific family of micro-architectures (physical hardware). The API Layer is a software abstraction layer wrapping ARC implementation details. API presents what kinds of logical services are provided and how they are grouped together and represented as interfaces. ASP (Application Spe-

cific Programmable) provides a group of application domain-specific services directly available to users. The API layer is the most useful layer among the three levels providing programmable and interactive interface for upper layer clients and applications [5][6].

2.2 QoS Offered By Embedded Platforms and Networks

QoS requirements specify not what the system does (provides services), but how the system satisfies its client requests while doing what it does [20]. The QoS relationship between the requester and the provider can be viewed from two aspects [3]:

- From Client/Server (horizontal) relationship: in which case a client specifies the required QoS and the server specifies the offered QoS for a negotiated contract.
- From an abstract/concrete (vertical) relationship: in which case the relationship is seen in a layered architecture . The MDA approach that we propose is related with this second aspect.

Considering the embedded networks, the two major categories of QoS mechanisms in Link Layer networks are **Reservation** and **Priority**. In reservation, network resources are allocated based on signaled requests originating from applications. Several parameters are used to define the reservation requirement and provision. Signaling messages are used to exchange such parameters. In prioritization (CAN, I2C), exchanged packets or frames are usually associated with a priority value that defines the handling in relation to other priorities. Several mechanisms for providing QoS exist in both categories. For example Bluetooth and IrDA use different **reservation** mechanisms. This work specifically focuses on the reservation and connection oriented category of the networks.

3 The Proposed MDA Approach

In enterprise MDA, the major focus is on modeling and transformation of functional elements and interfaces of applications from a more abstract to a more refined form, which does not consider the QoS aspects. We argue that such an MDA process is not generally suitable in the current and future embeddable communicable applications. Most of the application models must identify the behavior of their execution environment specially concerning QoS. More specifically, platform models in the embedded system development methodology greatly influence application models. The major concern is how to model the applications in order to use specific environments efficiently. Therefore, the model of the applications usually follows the model of the execution environment or is made along with the design of that specific environment (Co-design). Hence, unlike enterprise systems, the MDA approach for embedded systems in general should be based on the models of the platforms and their abstraction instead of application models and their refinement. The notion of “Abstract platform” [19], tailored with the MDA methodology will leverage the current challenges and visions in the embeddable communicable applications development.

Therefore what we propose is a model driven platform based (resource oriented) and QoS aware approach for embeddable communicable applications. This way the PIM of the platforms will be an abstract model that can be used within the model of the applications. Upon implementation the abstract platform will be mapped with a specific platform through a mapping layer. The mapping layer can target a number of different concrete platforms as shown in Fig 2.

3.1 Analysis of the Embedded Networks

Based on the analysis we have made on the reservation-based networks, the modeling elements are identified to be similar except the QoS expression and mechanism. Therefore, we present here the general model elements.
The objects (entities) identified are:

Channel/Connection: this refers to the channel identified with two endpoints on peers.

Event: every message exchange is produced as an event, which invokes a corresponding operation. It has four types: **Request**, **Indication**, **Response** and **Confirmation**,

QoS_Spec: this represents the QoS constraint (**Offered QoS**) of the link layers.

Service Interface: represents the service entity through which clients interact with the layer.

Classes will be used to represent these four entities. Using the terminology and modeling artifacts in the UML profiles defined in [2] and [3], Channel/Connection is considered as a Resource and QoS_Spec is a QoSConstranit. The other elements are modeled using the standard UML concepts.

3.2 The Platform Independent Model (PIM)

With the MDA standard, the PIM should be semantically similar to the platform models [7]. Hence, it has to reflect the connection oriented and the reservation based nature of the networks. The applications implemented in this network domain are aware of the reservation based and connection oriented nature of the networks. But their design and implementation will be independent of a specific network interface. Therefore the PIM concepts are based on abstract representations of platform specific characteristics. The essential model elements that the PIM must include in an abstract manner are the same as those of the specific platforms, except the QoS expression for which we propose a generic expression named **Flow_Spec** that can represent a reservation request. The **Flow_Spec** is an entity for the QoS as a generic reservation request specification taken from the flow specification proposed in [14] which is a **Token Bucket** based specification. It has been enhanced in [22] and further by Internet Engineering Task Force (IETF) for use in Internet reservation services [22][23]. For the QoS specification at the PIM level, the flow-based approach is selected for a number of reasons: First it is a closer approach to networks and in particular it is more appropriate for the connection oriented and reservation based networks consid-

ered in this work. Initially it was proposed for the Internet community. It is also a widely used model to quantitatively specify application requirements on a network. Second, it is more declarative than showing more technical details, which makes it appropriate for a PIM level specification according to the MDA standard . Third, its specification does not target a specific network protocol and reservation mechanism [12]21. This opens the possibility of mapping to many different specific implementations. Fourth, we believe that it is the most appropriate specification that can satisfy both requirements of a PIM stated in [15], i.e. platform independence and mappablity towards concrete platforms. We argue that this type of PIM specification can be transformed to Bluetooth, IrDA and other reservation based networks.

3.3 The Mapping Model

This section presents the detailed version of the proposed MDA approach as shown in Fig 2. The transformation between the PIM and the target network model (PM) is made through the intermediate mapping layer forming a PSM. This will meet the objective of MDA in that communicable embedded applications can be designed and implemented without the concern of the peculiar characteristics of the used network. For simplifying the model, the two concepts, i.e., the Functional Service and the QoS are separated into two groups (packages) as shown in Fig 2.

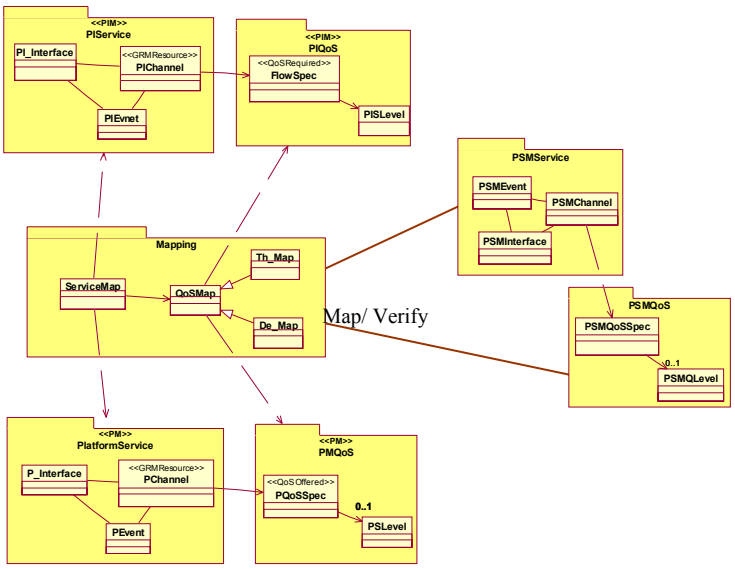


Fig. 2. The proposed MDA approach in UML.

3.5 The Mapping Strategy

The mapping layer will have two parts namely **ServiceMap,** responsible for mapping the functional service interface of the PIM with the PM and **QoSMap**, responsible for transforming the QoS modeling elements. It has two subtypes related to Throughput mappings (Th_Map) and latency mappings (De_Map).

Since the functional service mapping is relatively easier and is not our major focus area, we present here the most important part of the mapping layer, i.e., the QoS mapping. For the QoS mapping, the predictability nature of the specifications is considered. We have divided the QoS mapping strategy into three categories:

1. **Service Level Mapping**: This is done by using the semantics of the three levels at the PIM QoS. (**Guaranteed**, **Controlled Load** and **Best Effort**). Appropriate interpretations of the meanings in each specific network will be identified.
2. **Service Level Mapping**: This is done by using the semantics of the three levels at the PIM QoS. (**Guaranteed**, **Controlled Load** and **Best Effort**). Appropriate interpretations of the meanings in each specific network will be identified.
3. **Throughput Related Mapping:** for this case, we used a concept of Maximum Transmission Boundary (**MTB**) to determine the maximum amount of data bytes transferred within a period of time, based on the parameters for both PIM and PM. Hence, we must have:

$$\mathbf{MTB_{PIM} <= MTB_{PM}}\text{ , where } MTB_{PIM} = \min(B + r*T, M + P*T) \quad (1)$$

   where B=Bucket size, r = Token rate, T = a time interval for the flow, p = peak rate, and M = Maximum Transmission Unit [16][25]. Similarly, the corresponding value for the PM can be calculated from appropriate parameters.

A. **Latency Related Mapping:** this is done using the explicit specification at the PIM level and estimated from appropriate parameters at the PM level with the following relationship:

$$Latency_{PIM} >= Latency_{PM} \quad (2)$$

3.6 Procedures Used by the Mapping Layer

The mapping layer uses two procedures to link application requests with the underlying network level provisions. The **Map/Transform** procedure transforms and maps parameters from PIM to PM or vise versa. The **Verification** procedure verifies the Required/Offered relationship holds. Based on the requested service level appropriate action will be taken. If Guaranteed, requirements must be satisfied. If Controlled Load, requirements are flexible (negotiable), and if Best Effort, any value offered is accepted.

4 Applicability of the Approach

In this section, we present the applicability of our approach for the IrDA specific platform. We are forced to limit to the case of IrDA only due to page restrictions.
IrDA Platform Model: the IrDA link layer services are presented through the Link Management Protocol (IrLMP) layer which has a relatively similar purpose but slightly different functional services as the Bluetooth L2CAP layer. It also provides a connection-oriented service with a set of parameters for the level of QoS it provides to its clients.

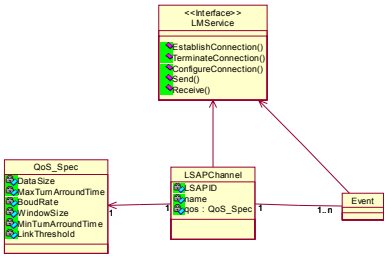


Fig. 3. The IrDA link layer Platform Model.

Since there is a difference in the parameters used to define the QoS provision, an indirect procedure is performed for mapping the PIM with the IrDA PM. The IrDA QoS parameters are defined in [24]. The mapping relationship is shown in Fig 4.

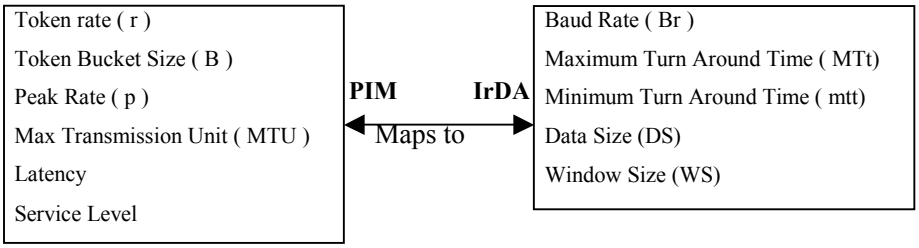


Fig. 4. Mapping the QoS parameters of the PIM and IrDA.

4.1 Mapping the Service Level

In the IrDA specification, there is no distinct expression for service levels. The most common scenario is that for the communication to begin between two application ends, both must agree on the exchanged QoS parameters. The negotiation values are distinct enumerated values. Furthermore, the two ends must honor the agreed upon values throughout the lifetime of the link. However, the PIM level specification takes the three service levels. Hence, the most appropriate accommodation in the link layer would be as follows.
If Guaranteed service level is requested, then strict parameters must be calculated and negotiated with the target IrDA link. The mapping layer then verifies the two values and decides on the success or failure. If “Best Effort” or “Controlled Load” service

levels are requested, there will be a possibility that the Offered values can override the required values if they do not agree.

4.2 Throughput Related Mapping

As an example we show Throughput related mapping. The major parameters that determine the MTB limit for IrDA are the **Br** and **MTt**. However, the actual limit is set from DS and WS within the MTB limit. Hence, we take: $\mathbf{MTB_{IrDA} <= Br * MTt}$
Since $MTB_{PIM} <= MTB_{IrDA}$, must hold and MTB_{PIM} = Min (B + r* MTt, MTU + p* MTt) , (Taking MTB_{PIM} = B + MTt * R and MTB_{IrDA} = MTt * Br), we have:

B+ MTt * R <= MTt * Br, then

$$Br \geq \frac{B + MTt * r}{MTt} \tag{3}$$

We used the MTt (Maximum Turn Around Time) of IrDA as an interval, because its value determines the brief break intervals the sender makes between each continuous burst of data flow, handing the link to the other device.
If Guaranteed level is requested by application (PIM), then the expression **Br>= r**, should be verified.
If Best Eeffort or Controlled Load, then the provided (Br) can override the required (r or p) and used for reverse calculation, and we have:

$$r \leq \frac{B - MTt * Br}{MTt} \quad \textbf{Or} \quad p \leq \frac{M - MTt * Br}{MTt}, \text{ and } \mathbf{p >= r} \tag{4}$$

Similarly, the mapping and the verification mechanism can be made for the **latency** related mapping.

5 Related Works

5.1 MDA for SoC

An MDA approach for System on Chip Design (SoC) methodology of embedded systems development has been addressed by the work in [15]. This approach is more appropriate for systems dedicated to specific tasks such as signal processing so that the functionality can be modeled for both the hardware and the software with the co-design methodology. Moreover, it does not consider interconnection between remote entities and how communication protocols and their variability are handled. In addition, it takes the hardware architecture as only the machine elements such as buses and chips and not the API level abstractions.

5.2 Network Protocol Modeling with UML

A first attempt has been made by Sekaran [16] for modeling a data link layer protocol specifically the L2CAP layer of Bluetooth. However, the major drawback of this work is that it does not consider the QoS provided at the link layer. It has only considered the functional services of the layer. Another similar work is that made by Thramboulidis and Mikiroyannidis [17] for modeling the TCP. However, the QoS issues have not been included in the model. These two works have shown that object oriented modeling and implementation of communication protocols is possible with its inherent benefits although slight performance penalty is expected.

5.3 Quality of Service Modeling Approaches

Quality of Service modeling is among the important issues addressed by different researches works recently. Several approaches for modeling QoS have been proposed [18],[19]. However, they do not address specific domains that are closely associated with QoS such as networks and embedded systems. Moreover, most of the concepts they have introduced are incorporated with the UML profiles discussed previously. In [20], Aagedal presents the concept of orthogonal separation between the QoS specification and the functionality specification of a system. Furthermore, it has shown how to link QoS aspect models with functional elements of models called computational elements such as Actor, Component, Interface, Node, Object, Subsystem, Use case, and Use case instance. But it does not use the MDA concepts such as Platform Independent Modeling, Platform Specific Modeling and Transformations.

6 Conclusion

In this paper, we have shown a possible adaptation of the MDA towards a QoS aware and resource oriented application development for the embedded systems domain. The major focus has been the communication subsystem of embedded platforms where the variability in the reservation based networks can be handled in a formal model based process. The Required/Provided relationship between applications and networks has been represented with the PIM and PM perspectives of the MDA and a possible mapping layer that can transform the application level requests to network level provisions. The applicability case study for IrDA has also shown that the PIM QoS can be transformed and also verified with specific network QoS. We believe that this way the concerns of application level modeling and implementation could be separated from the platform level service specification as two different concerns of development in this domain. In addition, we believe that the applicability of the mapping can work for other reservation based networks such as HiPERLAN2 even if it is not initially intended for embedded systems. In our work, we used only parameters that are used to define the performance requirements and provisions. In the real case, other factors such as the overhead imposed by the mapping layer should be considered.

References

1. ARTIST- Adaptive real-time systems for quality of service management- Roadmaps for Research (Draft) IST-2001-34820, 2003, pp 250-263, official site: http://www.artist-embedded.org/, visited on Jan 20, 2005.
2. OMG, UML Profile for Modeling Quality of Service and Fault Tolerance
3. OMG, UML Profile for Schedulability,Performance, and Time Specification:
4. Chen, R., Sgroi, M., Lavagno, L., Martin, G., Sangiovanni-Vincentelli, A., Rabaey, J., "Embedded Systems Design Using UML and Platforms", System Specification and Design Languages (Forum Design Languages 2002), CHDL Series, Kluwer, 2003.
5. Alberto Sangiovanni Vincentelli. Defining Platform-based Design. EEDesign of EETimes, February 2002.
6. Grant Martin, UML for Embedded systems specification and Design (IEEE document): http://ieeexplore.ieee.org/iel5/7834/21541/00998386.pdf, visited on Feb 15, 2005
7. Model-driven architecture - a technical perspective. Technical Report ORMSC/2001-07-01, Object Management Group, 2001. Online: http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01, visited on Feb 15, 2005
8. Bluetooth SIG, The Bluetooth Specification Document, available at www.bluetooth.com/pdf/Bluetooth_11_Specifications_Book.pdf
9. Patrick J. Megowan, David W. Suvak, Charles D. Knutson IrDA Infrared Communications, available at: An Overview: http://www.web-ee.com/primers/files/irda.pdf.
10. Robert Bosch, CAN Specification, Version 2.0, 1991, www.semiconductors.bosch.de/pdf/can2spec.pdf, visited on. Feb 25, 2005.
11. The I2C Specification, Version 2.1, January 2000: www.semiconductors.philips.com/ acrobat/literature/9398/39340011.pdf, visited on May 2, 2005
12. Partridge C., "A Proposed Flow Specification", Internet Engineering Task Force, Request for Comments: 1363, available at: http://www.ietf.org/rfc/rfc1363.txt, September 1992,
13. Javier Muñoz, Model Driven Development of Pervasive Systems, http://www.di.uminho.pt/~mompes04/Papers/Munoz.pdf, visited on Jan 15, 2005.
14. P.Boulet, J.L.Dekeyser, C.Dumoulin, and P.Marquet. MDA for SoC embedded systems design, intensive signal processing experiment. FDL03, 2003.
15. Jo˜ao Paulo Almeida et. al., Handling QoS in MDA: A Discussion on Availability and Dynamic Reconfiguration, CTIT Technical Report TR–CTIT–03–27, Univ. of Twente.
16. Sekaran K.C., Development of a Link layer protocol using UML, Proceedings of IEEE international Conference on Computer Networks and Mobile Computing, October 2001.
17. K. Thramboulidis, A. and A. Mikiroyannidis, Using UML for the Design of Communication protocols: The TCP Case study, 11th International Conference on Software Telecommunicationand Computer Networks, October 7-10, 2003.
18. Frolund, S. ; Koistinen, J.: QML: A Language for Quality of Service Specification / HP Labs. 1998 (TR-98-10). Technical report.
19. W. Torben, Model-Driven Development of QoS Enabled Distributed Applications, University of Electronics and Information Technology, Berlin, PhD thesis 2004.
20. Jan Øyvind Aagedal and Earl F. Ecklund, Jr., Modelling QoS: Towards a UML Profile: Presented at Fifth International Conference on the Unified Modeling Language -the Language and its applications (October 2002).
21. S. Shenker, J. Wroclawski, Network Element Service specification Template, RFC 2216, 1997: www.rfc-archive.org/getrfc.php?rfc=2216, visited on Jun 20, 2005.
22. Specification of Guaranteed Quality of Service (RFC 2212), 1997.
23. Specification of the Controlled Load Service (RFC 2211) IETF.
24. Andy Seaborne et.al., Infrared Data Association Link Management Protocol Specification:, Version 1.2, Jan 23, 1996

A Practical Experience on Model-driven Heterogeneous Systems Integration

Antonio Estévez[1], José D. García[1], Javier Padrón[1], Carlos López[1], Marko Txopitea[2], Beatriz Alustiza[3], José L. Roda[4]

[1]Open Canarias, SL, Elías Ramos González, 4, ofc. 304, S/C de Tenerife, 38001 España
info@opencanarias.com
[2]Open Norte, S.L., Madariaga Etorbidea, 1 – 4. Ezkerra, 48014 Bilbao, España
opennorte@opennorte.com
[3]IZFE , S.A., Pinares Plaza, 1 – 4. solairua, 20001 Donostia – San Sebastián, España
idazkari@gipuzkoa.net
[4]ULL, Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna, La Laguna, España
jlroda@ull.es

Abstract. The integration of heterogeneous systems is usually a complex task. In this study we present a strategy which can be followed for the integration of a framework based on Struts and J2EE, the transactional system CICS and the document manager FileNet. The principal aim of the project was to redefine the work methodology of the developers in order to improve productivity. Following model-based development strategies, especially MDA, a single framework for the three environments has been developed. Independent metamodels were created for each one of the environments, which finally led to a flexible, open and unified metamodel. The developer could then increase his productivity by abstracting from the particular implementation details related to each environment, and putting his efforts in creating a business model that is able to represent the new system.

1 Introduction

Most large business corporations and concerns use frameworks and heterogeneous tools in the running of their systems. Most of these systems need to integrate several architectures, technologies and information systems. At the moment, there are few consolidated solutions to solve these problems at a reasonable cost, which as well as being in house problems, will also depend on the technologies used to solve them.

The latest leanings have been towards strategies based on the Model-Driven Software Development (MDSD [13]), and more especially MDA [12] as a possible solution to most of the existing problems. Through MDA strategies, businesses can make sure that their business plans remain valid, independent from the frequent changes in the technology involved. Examples of the use of these kinds of strategies can be found in the following references: [3], [8].

In this paper we describe a general methodology for the integration of complex systems, based on the fundamental principles of MDSD, and applied in a real corporative environment. The principal objective of this project has been to put into

place a system of high productivity in order to develop J2EE [7] applications, which can interoperate with transactional systems such as CICS [4] and with content managers such as FileNet [5].

In part 2, the technological area is described along with the different platforms that have to be integrated. Then, we outline the development of three practical cases, using the selected platforms. The results of the application of the methodology, conclusions, and areas of future study will conclude this work.

2 Platforms Used in the Project

The Foral Society for Information Technology, pertaining to the Foral Department of Gipuzkoa (IZFE) has established and maintain an IT zone, with machines and servers made up from an IBM mainframe as well as more than 130 Windows, Unix and GNU/Linux servers, which are utilised by the Foral Department as well as the Town Councils of Gipuzkoa. IZFE is responsible for more than 90 new developments each year and at the moment has more than 300 applications up and running in a state of permanent evolution, with users as diverse as the Tax Office, the Departments of Transport, Culture and Youth, the Social Services departments, the Emergency services, as well as the Innovation. The number of persons working directly on these development projects has reached 165, without counting those collaborating within the closed environment of suppliers and providers.

The technologies used by IZFE in relation to this study are principally the Websphere Application Server for z/OS, version 5.1. The related database is DB2 Server for z/OS, version 7.1.0.

Of the technologies that we would like to integrate in the project, we would include the file manager FileNet, version 3.0, the transactional server CICS Transaction Server for z/OS, version 2.3 and the IZFE framework based on Struts [17] which allows for the development of J2EE applications.

This varied and complex group of platforms conform to the ideal scenario for the development of this project and we can apply the methodology and different integration strategies for the development of efficient software.

3 Methodology Used

Given the variety and complexity of the surroundings in which we worked, it was decided to use a bottom up methodology, beginning with the most specific aspects leading to generalizations and aspects in common. We thus planned a series of repeated tasks, differentiated by the technologies used in the study. At the end of these tasks, we proceeded to the integration of the different technologies, so as to link together with a usable common integrated model. Figure 1 shows the sequence of these tasks as well as the activities involved in each of them.

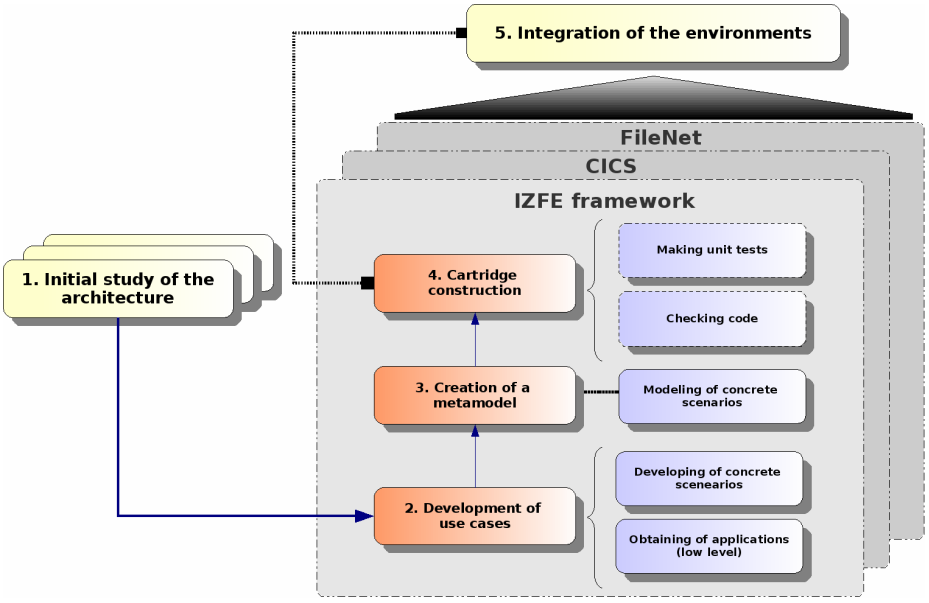


Fig. 1. Arrangement of the tasks with the methodology used.

Task 1. Initial study of the technology and architecture.
At this stage all the available information is obtained and studied. In order to do this, the IZFE is asked for all the relative information concerning his technological set-up that pertains to this project. All this information is checked, validated and developed. As far as possible, we try to make the study as near as possible to real life circumstances.

Task 2. Development of use cases.
At this stage, different applications (codes) are obtained for analysis and to define the functions required. As a result, a series of concrete scenarios should be developed, which comply in the most part with the functional requirements which have been analysed. To obtain programmes, we need wide ranging programmes to cover the technology that has been put in. These applications should for the most part cover the intrinsic necessities of the IZFE, covering the most common work tasks. The programmes are analysed and verified at a trial stage (with those technologies that already exist). These programmes are considered separately and recodified in order to easily create a metamodel, identifying structures and components which have the possibility of being considered separately, along with general concepts which previously had been modelled through the creation of templates. These templates were previously used for the automatic generation of codes in subsequent phases.

Task 3. Creation of a Metamodel.
The alignment of the project with MDSD and especially with MDA also involve the concept of UML Profiles [19], a specialised mechanism which is defined as being part of the same UML. The profiles can help shape specific aspects of a software system. The basic principle for obtaining each one of these profiles is to ascertain

generalisations between different programme languages, platforms and technologies, as well as to incorporate other relevant aspects related to the integration of inherited systems and applications.

Task 4. Cartridge Construction:
Having defined the functions and the metamodel, we can begin to construct the cartridge, whose function is to direct the working, compilation and packaging of the model exported in XMI [20]. In essence, a cartridge links the implementation of the UML profiles in a platform context with the programme language in which the code is generated, which in our case will be Java. This cartridge will contain a description where the profiles of each of the stereotypes are defined, and the corresponding template assigned. The new application is then checked through the IZFE's own technology.

These 4 tasks have been developed for the IZFE framework, CICS and FileNet areas, which we shall now explain in detail, and outline the cases where we had to customise the system in question.

4 Applying the Methodology

The development of the project followed a sequence through different technological environments. The IZFE framework was the first, as it was already identified as a key factor for the success of the project, as well as for its high level of complexity. The IZFE framework is a J2EE framework which runs on a Websphere Application Server. This is a server which is used extensively in the development of corporative web applications. The second environment considered was a transactional manager, identified as a CICS environment. In this environment there existed inherited processes and logic at a corporative level with a high strategic value. The third environment dealt with the development of an in company file manager, which was FileNet environment. In this environment there was a great quantity of high critical content, used in some areas of IZFE.

Each environment had definite tasks applied to them, using the methodology previously explained. The final phase of the project was to make a big effort to integrate all the environments into a common integrated metamodel.

4.1 IZFE Framework

Task 1. Study of the architecture. The IZFE framework is used for the creation of applications in a corporative business environment. Basically it is a fork of the Struts framework in the 1.1 version. The IZFE framework is divided into a series of subsystems, with the listener, control and presentation subsystems being of the first importance, as well as the business and the special security subsystems relevant in the corporative environment. Once the guides and reference information had been studied, an environment similar to that of IZFE was put in place.

Task 2. The development of use cases. Two applications were selected for the administration of the framework. These applications were tested and run in a simulated environment. Having these applications as a reference, the requirements could be defined for a new application and reengineering techniques were used in its implementation. During this phase, unitary components were identified, which could be used as parametric components in the metamodel.

Task 3. Creating the metamodel. The objective of the metamodel is to create a system with simplified architecture, which meets the requirements of the IZFE framework. To reach this simplified level, the components that the framework offered were mapped out to a model more inclined towards the MVC pattern [14], in which the domains are clearly defined and focussed on the functions of self contained web applications. With this simplification, we managed to reduce the elements of the MDA architecture which should be in place in order to create a more integrated application. It also permits a better distribution of the work needed to be done in the specialised areas, dividing the knowledge between different people that made up the team. The following domains and /or layers were defined: the initialisation, view, business logic, and persistence domains.

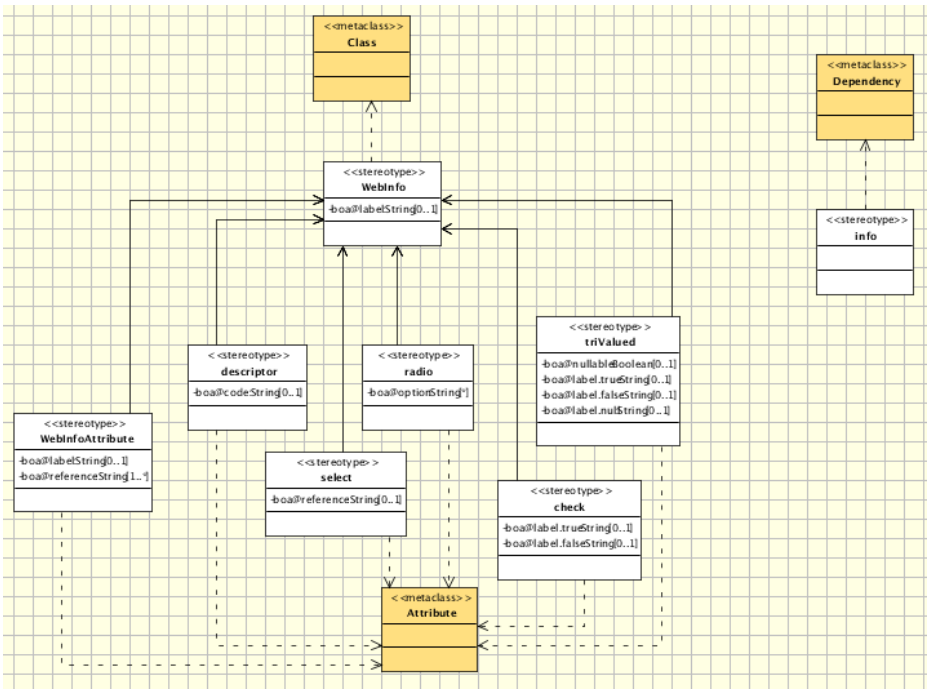


Fig. 2. Sample piece of the IZFE framework metamodel.

Task 4. Cartridge Construction. At this stage the objective proposed was the 100% code generation. This considerably increased the complexity of the problem, above all in the definition of the business logic. In order to reach this objective, state diagrams were used, incorporating into these states action semantics [1] which are described in

the specifications 1.5 of the UML. To reach this approximation, Action Specification Language was employed (ASL) [2], and with certain modifications a grammar and a parser were developed, using a compiler from the SableCC [15] compiler. In this way a cartridge which generated completely automatic applications was obtained. Now the IZFE, instead of programming these interfaces directly, uses the metamodels defined in the UML in order to represent their needs graphically. The system is capable of automatically generating codes from these diagrams.

4.2 CICS Environment

Task 1. Architecture study. The objective to be reached in this environment is the running of complex programmes hosted in CICS through J2EE components. An exhaustive study of this area was needed, due to the non-existence of any previous development programmes with the requirements specified in this study. Two key problems were identified, the communication with the EIS, and the formatting of types between domains.

Task 2. Development of existing cases. To solve the communication problem, the CICS ECI Recorder Adapter was used, put through the CTG (IBM CICS Transaction Gateway). The second problem identified in Task 1 was solved using the JRIO [10] library. Finally the minimum functions required were obtained through unitary tests in order to validate the solution.

Task 3. Creating the metamodel. The metamodel was developed by identifying the general functional components, and parametricizing the minimum information which is needed by the models in order for the previous correct generation to be attained. All this has been effected using as a reference the use cases which were put together in previous times.

Task 4. Cartridge Construction. Finally the cartridge was implemented, due to the fact that the code was automatically generated in the context of the platform, which included a small unitary test. This cartridge contains a descriptor where the profiles are defined with each of the stereotypes assigned to the corresponding templates. Beforehand, the generated systems were checked. In this way, and using a generation motor, the IZFE can describe the model graphically through simple UML diagrams, and generate 100% the code needed for the connection of the transactional manager.

4.3 FileNet Framework

Task 1. Architecture study. FileNet is a document manager with the added functions of workflow and with its own framework based on Struts. It has an API for Java which allows access to practically all of its functions. IZFE has developed and maintains a small simplified API which makes easier the running of the contents of the organisation's internal uses.

Task 2. Development of existing cases. Two extracts from two different applications were selected which made use of the API of IZFE. Based on the examples provided and using inverse reengineering, the common functions were extracted in real scenarios. Finally a series of unitary tests were made in the IZFE's environment.

Task 3. Creating the metamodel. The metamodel was developed by identifying functional components that needed to be generalised, and then parametricizing the minimum information needed to the models for their correct working. All this was carried out using past existing cases as a point of reference.

Task 4. Cartridge Creation. For the construction of the cartridge each one of the stereotypes were mapped out to the units of generation. A template was defined for each unit of generation, which allows for a generator motor for the creation of codes. The use of a cartridge allows, through the definition of UML diagrams, for the 100% generation of an access code of the resource contents defined in the corresponding document manager.

4.4 Unification of the Metamodels

The last stage defined in point 3 corresponds to then integration of the different environments. This can be done at a web level using the IZFE framework.

Initially integration with CICS was planned. A metamodel was obtained and a cartridge for the modelling independent of an actual programme, in which some entry parameters were effected and some exit parameters were obtained. Beforehand, this function was incorporated as another element, to be integrated into the metamodel used in the design of the models which are generated for the IZFE framework.

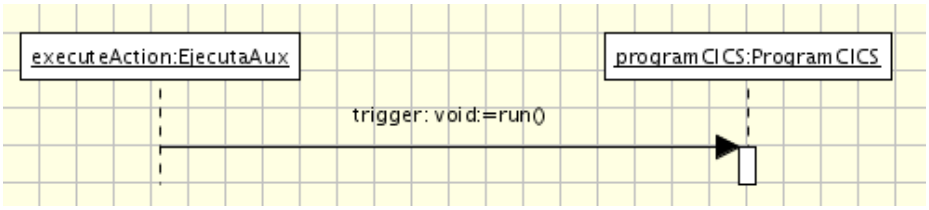


Fig. 3. Sequence diagram sample about CICS integration in IZFE framework in the view domain.

Thus, two types of integration were accommodated in the IZFE framework metamodel. One integration in the presentation domain, which permitted the use of the CICS programme using a web form. A second integration was in the business logic domain. This last integration was effected using state diagrams.

The integration with FileNet was approached in a similar way to the CICS. Once the metamodel and cartridge have been created, they can be used independently and in isolation, and be used along with the integration of the IZFE framework metamodel. This implies an enlargement, not only in the persistence domain (resource persistence), but also in the presentation domain. So a maintenance

environment has to be established, as well as the running of a general resource manager, which in turn permits in a simple form the creation, modification, cleaning and search for FileNet resources.

A two-step strategy for problem solving was used. The first step a complete system was generated using a traditional model for IZFE framework. Beforehand, a system generated to create templates was used, with a high abstract level, which in turn allowed for the definition of a series of stereotypes, which simplifies even more the definition of the integration models with FileNet. This strategy proved successful due to the few cases of variation in the initial requirements in the use of the FileNet resources.

5 Results Obtained

Using this system, an architect could construct a complete application, designing the adequate models based on UML with profiles. In order to do this, it is not necessary to be an expert in J2EE, nor in the IZFE framework, nor in CICS or FileNet; it is enough just to have a basic knowledge of these technologies and in UML.

With the correct modelling, the engine on which the project is based is capable of generating the total structure and codes needed for the start up and development of a complete and full application of a J2EE server, as in the case of a Websphere Application Server (WAS). This newly formed application is totally compatible with the corporative IZFE framework, and could be used, in the business layer with functions stored in CICS systems, or in the persistence layer, with defined resources in the document FileNet database. All this can be achieved without inserting even one line of code, and without being an expert in the technologies employed, solely by simply correctly modelling through the UML diagrams.

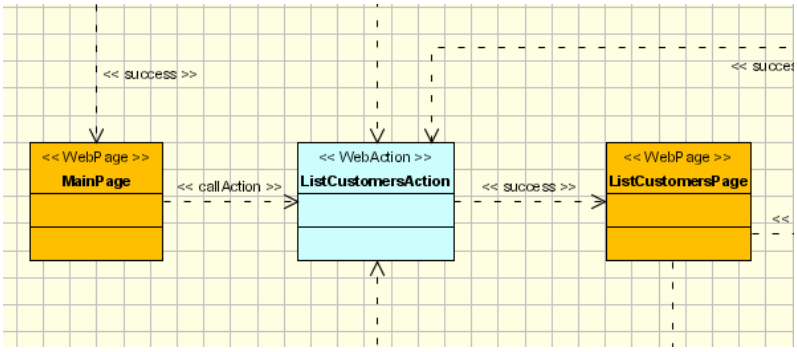


Fig. 4. Example of an actual model in the view domain.

It should also be noted that the persistence area of the business model in related databases in the IZFE framework gives a free hand for those programmers who would like to propose solutions that they feel are adequate to any given situation. We therefore propose the use of Hibernate [6] for the running of this project as a viable and efficient solution for the persistence of these objects with whichever database that

is being utilised. (in our case, DB2). This component was added, and then modelled and run through an adequate cartridge. For the connection to the EIS, in this case the CICS, we made use of the literature provided by JCA [9], where numerous references, documents and up to date texts were consulted. And for the converting of data to Cobol and vice versa, a crucial function in this area, texts and classes provided by JRIO were used. The persistence of resources was also modelled using a FileNet content manager. The metamodel had been sufficiently abstracted for it to be used generally for other content managers, by solely modifying the cartridge.

Finally, it must be noted that on the completion of this project it was possible to integrate all the previous metamodels into a common unified metamodel. With this, the complete integration of specific environments has been reached which are completely heterogeneous within a unified, efficient and ordered model, which in turn allows for the development of new systems. Anyone developing these systems would find a framework based on UML, with highly defined profiles in all three platforms, which can lead to a higher level of abstraction, working independently from the technological aspects.

6 Conclusions

This new paradigm in systems creation represents a big change in the traditional way of working of the development teams in IZFE. A new methodology was embedded as well as new work practices, along with the planning needed in the management of change to the rapid adaptation of the new paradigm to take full advantage of the new environment. The modellers of the new systems should possess a high degree of knowledge of UML in order to work on the theory and creation of these systems.
Apart from the intrinsic advantages derived from a system based on an approximation of MDA, we can also note:

- The normalisation of the systems through UML models.
- The integration of heterogeneous systems, which hide the complexities of each of the technologies in question.
- The development of one system only based on the Web.
- The considerable rise in the quality of the systems developed, given that the codes generated had been exhaustively tested.
- The possibility of a rapid development of prototypes, which could be easily converted into systems and final applications.
- The improvement in the facility of implementing the persistence of the models, independent from the continual technological change and evolution.

7 Future Proposals

For the future, we are working towards the adaptation and maintenance of the cartridge that has been made, according to how the systems already integrated have evolved (IZFE framework CICS and FileNet), as well as other technologies. Other different cartridges can be generated for other programming languages apart from

Java (.NET for example), as well as the incorporation of other tools (Spring [16]) into the corporative framework, or the interaction with other systems different from those in place, which would mean a modification of the cartridges.

There is also the possibility of integrating the technology into portlets [11], a challenge for the domain of our application. The portlet provided by Struts is recommended in order to avoid any compatibility problems with the IZFE framework controller.

In the MDA environment, it is worth noting the emergence and use of new engines for code generation. In this case, a "translational" method has been used, where, apart from the templates included in the corresponding cartridge, we also managed to put the model designed into code. There also exists at the moment other methods known as "elaborational", where changes are made to models based on QVT [18]. This method has a great future within MDA architecture.

References

1. Action Semantics Revised Final Submission. OMG document ad/01-08-04.SL
2. ASL – The Action Specification Language Reference Manual. http://www.kc.com
3. Brunton R., Brutzman D., Drake D., Hieb M., Morse K.L., Pullen J.M., Tolk A. : "Using Web Services to Integrate Heterogeneous Simulations in a Grid Environment", Lecture Notes in Computer Science, Springer-Verlag Heidelberg (2004) pp. 835 – 847.
4. CICS – Customer Information Control System. http://www-306.ibm.com/software/htp/cics/
5. FileNet http://www.filenet.com P8 3.0.0 Documentation
6. Hibernate: http://www.hibernate.org
7. J2EE – Java 2 Platform, Enterprise Edition. http://java.sun.com/javaee/index.jsp
8. Jahnke, J.H., Wadsack, J.P. : "Towards Model-Driven Middleware Maintenance", Proc. of the OOPSLA 2002 Workshop on Generative Techniques in the context of Model-Driven Architecture, Seattle, USA., November 2002.
9. JCA – J2EE Connector Architecture. http://java.sun.com/j2ee/connector/
10. JRIO – Java Record I/O. http://www-03.ibm.com/servers/eserver/zseries/software/java/jrio/overview.html
11. JSR 168, portlet specification. http://www.jcp.org/en/jsr/detail?id=168
12. MDA – Model Driven Architecture. http://www.omg.org/mda/
13. MDSD – Model-Driven Software Development. http://www.mdsd.info/
14. MVC –Model View Controller pattern. http://java.sun.com/blueprints/patterns/MVC-detailed.html
15. SableCC Parser generator. http://sablecc.org
16. Spring framework. http://www.springframework.org
17. Struts Framework http://struts.apache.org/
18. QVT – Query Views Transformations. http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF_QVT
19. UML – Unified Modelling Language http://www.uml.org/
20. XMI – XML Metadata Interchange. http://www.omg.org/technology/documents/formal/xmi.htm

Author Index