Michel R.V. Chaudron (Ed.)

Models in Software Engineering

Workshops and Symposia at MODELS 2008
Toulouse, France, September 28 – October 3, 2008
Reports and Revised Selected Papers

Springer

Volume Editor

Michel R.V. Chaudron
University of Leiden
Faculty of Science
Leiden Institute of Advanced Computer Science
P.O. Box 9512, 2300 RA Leiden, The Netherlands
E-mail: chaudron@liacs.nl

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.2, D.3, I.6, K.6

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-642-01647-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-01647-9 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12607341 06/3180 5 4 3 2 1 0

Preface

Following the tradition of previous editions of the MODELS conference, many satellite events were organized in co-location with the MODELS conference in Toulouse in 2008: 12 workshops, 3 symposia, 9 tutorials, a poster session, and a tools exhibition. The selection of the workshops was organized by a Workshop Selection Committee, which consisted of the following experts:

– Michel R.V. Chaudron, Leiden University, The Netherlands (Chair)
– Jochen Küster, IBM Research Zurich, Switzerland
– Henry Muccini, University of L'Aquila, Italy
– Holger Giese, Hasso-Plattner-Institute, Germany
– Hans Vangheluwe, McGill University, Canada

Some workshops have been running for several years as MODELS satellite events, but each year some workshops end. Furthermore, there are always new developments, and hence there is room for new workshops. Therefore, the Workshop Selection Committee very much welcomes new proposals.

The workshops enabled groups of participants to exchange recent and/or preliminary results, to conduct intensive discussions, or to coordinate efforts between representatives of a technical community. They served as forums for lively discussion of innovative ideas, recent progress, or practical experience on model-driven engineering for specific aspects, specific problems, or domain-specific needs.

The three symposia this year were: the Doctoral Symposium, the Educators' Symposium, and the Research Projects Symposium. The Doctoral Symposium provided specific support for PhD students to discuss their work and receive guidance for the completion of their dissertation research. The Educators' Symposium addressed the question of how to educate students and practitioners to move from traditional thinking to an engineering approach based on models. The Research Projects Symposium was a showcase for research projects, as well as a forum where researchers from academia and industry and representatives of funding agencies could debate on technology transfer and trends in research projects.

These satellite-event proceedings were published after the conference and include summaries as well as revised versions of the best papers from the workshops, the Doctoral Symposium, the Educators' Symposium, and the Research Projects Symposium.

I would like to thank everyone involved in making the satellite events such a successful and inspiring experience.

January 2009 Michel R.V. Chaudron

Table of Contents

Model Based Architecting and Construction of Embedded Systems

Iulian Ober[1], Stefan Van Baelen[2], Susanne Graf[3], Mamoun Filali[4], Thomas Weigert[5], and Sébastien Gérard[6]

[1] University of Toulouse - IRIT, France
Iulian.Ober@irit.fr
[2] K.U.Leuven - DistriNet, Belgium
Stefan.VanBaelen@cs.kuleuven.be
[3] Université Joseph Fourier - CNRS - VERIMAG, France
Susanne.Graf@imag.fr
[4] University of Toulouse - CNRS - IRIT, France
Filali@irit.fr
[5] Missouri University of Science and Technology, USA
weigert@mst.edu
[6] CEA - LIST, France
Sebastien.Gerard@cea.fr

Abstract. This workshop brought together researchers and practitioners interested in model-based software engineering for real-time embedded systems, with a particular focus on the use of architecture description languages, domain-specific design and implementation languages, languages for capturing non-functional constraints, and component and system description languages. Ten presenters proposed contributions on model-based analysis, transformation and synthesis, as well as tools, applications and patterns. Three break-out groups discussed the transition from requirements to architecture, design languages, and platform (in)dependence. This report summarises the workshop results.

1 Introduction

The development of embedded systems with real-time and other constraints implies making specific architectural choices ensuring the satisfaction of critical non-functional constraints (in particular, related to real-time deadlines and platform parameters, such as energy consumption or memory footprint). Recently, there has been a growing interest in (1) using precise (preferably formal) domain-specific models for capturing dedicated architectural and non-functional information, and (2) using model-driven engineering (MDE) techniques for combining these models with platform independent functional models to obtain a running system. As such, MDE can be used as a means for developing analysis oriented specifications that, at the same time, represent the design model.

The MoDELS workshop on *"Model Based Architecting and Construction of Embedded Systems"* brought together researchers and practitioners interested in all aspects of model-based software engineering for real-time embedded systems. The participants discussed this subject at different levels, from modelling languages and related semantics to concrete application experiments, from model analysis techniques to model-based implementation and deployment. The workshop was attended by 61 registered participants hauling from 18 different countries, including six outside of Europe.

2 Reviewed Contributions

We received 16 submissions from 8 different countries, of which ten were accepted for presentation. A synopsis of each presentation is given below. Articles [4,10] are included in this workshop reader, while the other articles can be found in the workshop proceedings [1].

[2] proposes a method for estimating the power consumption of components in the AADL (Architecture Analysis and Design Language) component assembly model, once deployed onto components in the AADL target platform model.

[3] proposes Visual Timed Scenarios (VTS) as a graphical property specification language for AADL. An effective translation from VTS to Time Petri Nets (TPN) has been devised, which enables model-checking of properties expressed in VTS over AADL models using TPN-based tools.

[4] describes a general methodology and an associated tool for translating AADL and its annex behavior specification into the BIP (Behavior Interaction Priority) language. This allows simulation of systems specified in AADL and application of formal verification techniques developed for BIP to these systems.

[5] discusses the transformation of a global requirements model into a system design. It describes an algorithm that derives the local behaviors for each system component, including coordination messages between the different system components.

[6] shows how higher-order model composition can be employed for constructing scalable models. An approach based on model transformation is presented, defining basic transformation rules operating on the graph structures of actor models.

[7] demonstrates an approach for early cross toolkit development based on the ISE ADL and the MetaDSP framework. It is designed to cope with hardware design changes, like changes in the instruction set of target CPUs.

[11] Discusses advantages and drawbacks of UML and SysML for modeling RF systems, based on a case study of a UMTS transceiver.

[8] propose the use of a new notion of modeling patterns for arbitrating between the real-time system designer's needs of expressive power and the restrictions that must be imposed so that models can be amenable to static analysis.

[9] proposes an approach based on UML-MARTE that allows system modelling with separation of concerns between the functional software model and the execution platform resources as well as timing constraints. Temporal characteristics and timing constraints can be specified at different abstraction levels, and scheduling analysis techniques can be applied on the resulted model.

[10] presents a model-based integration environment which uses a graphical architecture description language (EsMoL) to pull together control design, code and configuration generation, platform-specific resimulation, and a number of other features useful for taming the heterogeneity inherent in safety-critical embedded control system designs.

3 Discussion of Breakout Sessions

After the presentations, the participants broke into three different groups, each one focusing on a particular subject of interest. The following summarizes the conclusions of each breakout session.

Transition from requirements to architecture. There is a need for formalisation of requirements, better traceability, structuring and partitioning, validation, verification, and enforcement of requirements. Too often design decisions are already made during requirements decomposition. Given a requirements specification, the next step is to guide the developers towards a good architecture. A manual approach can be followed, by which the developer defines proper components, makes architectural choices, and allocates requirements to system elements. The result is analyzed and incrementallhy improved until a suitable system design has been found. Alternatively, a semi-automatic approach could be used by defining the constraints on the system, generating solutions that satisfy these constraints, and adding more constraints in order to reduce the solution space. The key difficulty for this approach is to find suitable constraints to add, in order to avoid inadvertently eliminating possible solutions. One could leverage patterns for obtaining detailed architectures and designs. The key issue here is how to deal with the interaction between patterns, since a system often has to deal with many competing concerns and accompanying patterns. Finally, after the system has been designed it has to be verified. A model refinement approach can insure correctness by construction, but a key concern is whether such can be done in an incremental manner, and how to resolve the different conflicting aspects. Another verification approach is by model analysis and model checking. The key question here is to build a relevant abstraction/analysis model in order to verify the properties of interest.

Design Languages. Modeling languages are in wide-spread use today, at least in the prototyping phase, but we are facing a scalability problem and many conflicting demands on a modeling language: Modeling languages are necessary for expressing our understanding of the problem. Modeling languages should also be used for existing software. They should provide ways to go back and forth between designs and existing code. They should support consistent views between different levels of abstraction. Modeling languages cannot not be generic, but need to be domain-oriented. However, while for some domains the mapping between the formal model and its implementation is well understood, for other domains this has yet to be demonstrated. The use of modeling languages relies essentially on tools which are heavily dependent on version changes, and users should be aware of that. Version control is a real problem. Education is very important, since when people enter the work force it is often too late to teach new methods. Teaching formal methods, or at least teaching rigorous modelling, is a must.

Transformations and platform (in)dependence. This discussion group concentrated on the meaning of platform independence and on the types of transformations in which platform independent models can be involved. A model can only be independent from a specific understanding of what the platform is, and in order to achieve this independence it must include a model of the platform. Such a model could specify, for example, a specific model of computation, a set of services, a set of resources, or required QoS constraints. Only in the presence of such platform model can a model be qualified as platform independent, which means, independent with respect to a concrete platform which conforms to the constraints specified by the platform model. The platform model is often not explicit, which reduces the self-containment and the usability of models.

Acknowledgements. We thank all workshop participants for the lively discussions and the useful feedback we received. We thank the workshop program committee members for their helpful reviews. This workshop was supported by the ARTIST2 Network of Excellence on Embedded Systems Design (http://www.artist-embedded.org) and by the EUREKA-ITEA project SPICES (http://www.spices-itea.org).

References

1. Van Baelen, S., Ober, I., Graf, S., Filali, M., Weigert, T., Gérard, S. (eds.): ACES-MB 2008 Proceedings: First International Workshop on Model Based Architecting and Construction of Embedded Systems, IRIT, Toulouse, France (2008)
2. Senn, E., Laurent, J., Diguet, J.-P.: Multi-level power consumption modelling in the aadl design flow for dsp, gpp, and fpga. In: Van Baelen, et al. (eds.) [1], pp. 9–22
3. Monteverde, D., Olivero, A., Yovine, S., Braberman, V.: Vts-based specification and verification of behavioral properties of aadl models. In: Van Baelen, et al. (eds.) [1], pp. 23–37
4. Chkouri, M.Y., Robert, A., Bozga, M., Sifakis, J.: Translating aadl into bip - application to the verification of real-time systems. In: Van Baelen, et al. (eds.) [1], pp. 39–53
5. Bochmann, G.V.: Deriving component designs from global requirements. In: Van Baelen, et al. (eds.) [1], pp. 55–69
6. Feng, T.H., Lee, E.A.: Scalable models using model transformation. In: Van Baelen, et al. (eds.) [1], pp. 71–85
7. Pakulin, N., Rubanov, V.: Ise language: The adl for efficient development of cross toolkits. In: Van Baelen, et al. (eds.) [1], pp. 87–98
8. Bordin, M., Panunzio, M., Santamaria, C., Vardanega, T.: A reinterpretation of patterns to increase the expressive power of model-driven engineering approaches. In: Van Baelen, et al. (eds.) [1], pp. 145–158
9. Peraldi-Frati, M.-A., Sorel, Y.: From high-level modelling of time in marte to real-time scheduling analysis. In: Van Baelen, et al. (eds.) [1], pp. 129–143
10. Porter, J., Karsai, G., Völgyesi, P., Nine, H., Humke, P., Hemingway, G., Thibodeaux, R., Sztipanovits, J.: Towards model-based integration of tools and techniques for embedded control system design, verification, and implementation. In: Van Baelen, et al. (eds.) [1], pp. 99–113
11. Lafi, S., Champagne, R., Kouki, A.B., Belzile, J.: Modeling radio-frequency front-ends using sysml: A case study of a umts transceiver. In: Van Baelen, et al. (eds.) [1], pp. 115–128

Translating AADL into BIP - Application to the Verification of Real-Time Systems*

M. Yassin Chkouri, Anne Robert, Marius Bozga, and Joseph Sifakis

Verimag, Centre Equation - 2, avenue de Vignate 38610 GIERES

Abstract. This paper studies a general methodology and an associated tool for translating AADL (Architecture Analysis and Design Language) and annex behavior specification into the BIP (Behavior Interaction Priority) language. This allows simulation of systems specified in AADL and application to these systems of formal verification techniques developed for BIP, e.g. deadlock detection. We present a concise description of AADL and BIP followed by the presentation of the translation methodology illustrated by a Flight Computer example.

1 Introduction

AADL [5] is used to describe the structure of component-based systems as an assembly of software components mapped onto an execution platform. AADL is used to describe functional interfaces and performance-critical aspects of components. It is used to describe how components interact, and to describe the dynamic behavior of the runtime architecture by providing support for model operational modes and mode transitions. The language is designed to be extensible to accommodate analysis of runtime architectures.

An AADL specification describes the software, hardware, and system part of an embedded real-time system. Basically, an AADL specification is composed of components such as data, subprogram, threads, processes (the software side of a specification), processors, memory, devices and buses (the hardware side of a specification) and system (the system side of a specification).

The AADL specification language is designed to be used with analysis tools that support automatic generation of the source code needed to integrate the system components and build a system executable.

BIP [9] is a language for the description and composition of components as well as associated tools for analyzing models and generating code on a dedicated platform. The language provides a powerful mechanism for structuring interactions involving rendezvous and broadcast.

In order to demonstrate the feasibility of the BIP language and its runtime for the construction of real-time systems, several case studies were carried out such as an MPEG4 encoder [15], TinyOS [10], and DALA [8].

* This work is partially supported by the ITEA/Spices project as well as by the STIC-AmSud project TAPIOCA.

This paper provides a general methodology for translating AADL models into BIP models [4]. This allows simulation of systems specified in AADL and application to these systems of formal verification techniques developed for BIP, e.g. deadlock detection [11].

We use existing case studies [3,2] to validate the methodology. This paper is organized as follows. Section 2 gives an overview of AADL and annex behavior specification. Section 3 gives an overview of BIP. In section 4, we translate AADL components (software, hardware, system and annex behavior specification). We present our tool in Section 5. In section 6, we present a Flight Computer example. Conclusions close the article in Section 7.

2 Overview of AADL

2.1 Generalities

The SAE Architecture Analysis & Design Language (AADL) [5] is a textual and graphical language used to design and analyze the software and hardware architecture of performance-critical real-time systems. It plays a central role in several projects such as Topcased [7], OSATE [6], etc.

A system modelled in AADL consists of application software mapped to an execution platform. Data, subprograms, threads, and processes collectively represent application software. They are called *software components*. Processor, memory, bus, and device collectively represent the execution platform. They are called *execution platform components*. Execution platform components support the execution of threads, the storage of data and code, and the communication between threads. Systems are called *compositional components*. They permit software and execution platform components to be organized into hierarchical structures with well-defined interfaces. Operating systems may be represented either as properties of the execution platform or can be modelled as software components.

Components may be hierarchical, i.e. they my contain other components. In fact, an AADL description is almost always hierarchical, with the topmost component being an AADL system that contains, for example, processes and processors, where the processes contain threads and data, and so on.

Compared to other modeling languages, AADL defines low-level abstractions including hardware descriptions. These abstractions are more likely to help design a detailed model close to the final product.

2.2 AADL Components

In this section, we describe the fragment of AADL components, connections and annex behavior taken into account by our translation.

Software Components. AADL has the following categories of software components: subprogram, data, thread and process.

```
subprogram operation
    features
        A: in parameter integer;
        B: in parameter integer;
        result: out parameter integer;
end operation;
```

```
data Person
end Person;
data implementation Person.impl
   subcomponents
       Name : data string;
       Adress: data string;
       Age : data integer;
end Person.impl;
```

Fig. 1. Example of AADL subprogram and data

Subprogram : A subprogram component represents an execution entry-point in the source text. Subprograms can be called from threads and from other subprograms. These calls are handled sequentially by the threads. A subprogram call sequence is declared in other subprograms or thread implementations.

A subprogram type declaration contains *parameters* (in and out), out *event ports*, and out *event data ports*. A subprogram implementation contains *connections* subclause, a *subprogram calls* subclause, *annex behavior* subclause, and subprogram *property* associations. Figure 1 gives an example of a subprogram, that takes as input two integers A, B, and produces the result as output.

Data : The data component type represents a data type in the source text that defines a representation and interpretation for instances of data. A data implementation can contain *data* subcomponents, and data *property* associations. An example of data is given in Figure 1.

Thread : A thread represents a sequential flow of control that executes instructions within a binary image produced from source text. A thread always executes within a process. A scheduler manages the execution of a thread.

A thread type declaration contains ports such as *data port*, *event port*, and *event data port*, *subprogram* declarations, and *property* associations. A thread component implementation contains *data* declarations, a *calls* subclause, *annex behavior*, and thread *property* associations.

Threads can have properties. A property has a name, a type and a value. Properties are used to represent attributes and other characteristics, such as the *period*, *dispatch protocol*, and *deadline* of the threads, etc. Dispatch protocol is a property which defines the dispatch behavior for a thread. Four dispatch protocols are supported in AADL: *periodic*, *aperiodic*, *sporadic*, and *background.*

Figure 2 presents a thread component called sensor, that is a periodic thread with inter-arrival time of 20ms. This thread receives an integer data through port inp and sends an event through port outp.

Process : A process represents a virtual address space. Process components are an abstraction of software responsible for executing threads. Processes must contain at least one explicitly declared *thread* or thread group, and can contain a *connections* subclause, and a *properties* subclause. Figure 2 presents an example of process called Partition, that contains thread subcomponents and two types of connections (*data port* and *event port*) between threads.

```
thread sensor
    features
        inp : in data port integer;
        outp : out event port;
    properties
        Dispatch_protocol=>Periodic;
        Period => 20ms;
end sensor;
```

```
process Partition
end Partition;
process implementation Partition.Impl
  subcomponents
      Sensor_A : thread Sensor_Thread.A;
      Data_Fusion: thread Fusion.Impl;
      Alrm : thread Alrm_Thread.Impl;
  connections
      data port
        Sensor_A.outp->Data_Fusion.inpA;
      event port
        Sensor_A.launch->Alrm.launch_A;
end Partition.Impl;
```

Fig. 2. Example of AADL thread and process

Hardware Components. Execution platform components represent hardware and software that is capable of scheduling threads, interfacing with an external environment, and performing communication for application system connections. We consider two types of hardware components: processors and devices.

Processor : AADL processor components are an abstraction of hardware and software that is responsible for scheduling and executing threads. In other words, a processor may include functionality provided by operating systems.

Device : A device component represents an execution platform component that interfaces with the external environment. A device can interact with application software components through their ports.

Systems. A system is the toplevel component of the AADL hierarchy of components. A system component represents a composite component as an assembly of software and execution platform components. All subcomponents of a system are considered to be contained in that system. We present an example of system:

```
system Platform
end Platform;
system implementation Platform.Impl
  subcomponents
      Part : process Partition.Impl;
      p : processor myProcessor ;
      ...
end Platform.Impl;
```

Annex Behavior Specification. Behavior specifications [1] can be attached to AADL model elements using an annex. The behavioral annex describes a transition system attached to subprograms and threads. Behavioral specifications are defined by the following grammar:

```
annex behavior_specification {**
        <state variables>? <initialization>? <states>? <transitions>?
**};
```

– *State variables* section declares typed identifiers. They must be initialized in the *initialization* section.
– *States* section declares automaton states.
– *Transitions* section defines transitions from a source state to a destination state. The transition can be guarded with events or boolean conditions. An action part can be attached to a transition.

Connections. A *connection* is a linkage that represents communication of data and control between components. This can be the transmission of control and data between ports of different threads or between threads and processor or device components. There are two types of connections: port connections, and parameter connections.

Port connection: Port connections represent transfer of data and control between two concurrently executing components. There are three types of port connections: *event*, *data* and *event data*.

Parameter connection: represent flow of data between the parameters of a sequence of subprogram calls in a thread.

3 The BIP Component Framework

BIP (Behavior Interaction Priority) is a framework for modeling heterogeneous real-time components [9]. The BIP component model is the superposition of three layers: the lower layer describes the behavior of a component as a set of transitions (i.e. a finite state automaton extended with data); the intermediate layer includes connectors describing the interactions between transitions of the layer underneath; the upper layer consists of a set of priority rules used to describe scheduling policies for interactions. Such a layering offers a clear separation between component behavior and structure of a system (interactions and priorities).

The BIP framework consists of a language and a toolset including a frontend for editing and parsing BIP programs and a dedicated platform for model validation. The platform consists of an Engine and software infrastructure for executing models. It allows state space exploration and provides access to model-checking tools of the IF toolset [13] such as Aldebaran [12], as well as the D-Finder tool [11]. This permits to validate BIP models and ensure that they meet properties such as deadlock-freedom, state invariants [11] and schedulability. The BIP language allows hierarchical construction [14] of composite components from atomic ones by using connectors and priorities.

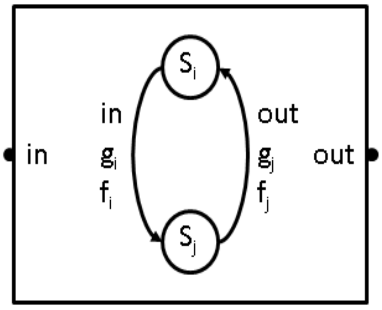


Fig. 3. BIP Atomic Component

An *atomic* component consists of a set of *ports* used for the synchronization with other components, a set of transitions and a set of local variables. Transitions describe the behavior of the component. They are represented as a labeled relation between *control states.* A transition is labeled with a port p, a guard g and a function f written in C. The guard g is a boolean expression on local variables and the function f is a block of C code. When g is true, an interaction involving p may occur, in which case f is executed. The interactions between components are specified by connectors.

Figure 3 shows an atomic component with two control states S_i and S_j, ports *in* and *out*, and corresponding transitions guarded by guard g_i and g_j .

Interactions between components are specified by *connectors.* A connector is a list of ports of atomic components which may interact. To determine the interactions of a connector, its ports have the synchronization attributes *trigger* or *synchron*, represented graphically by a triangle and a bullet, respectively. A connector defines a set of interactions defined by the following rules:

- If all the ports of a connector are synchrons then synchronization is by *rendezvous.* That is, only one interaction is possible, the interaction including all the ports of the connector.
- If a connector has one trigger port then synchronization is by *broadcast.* That is, the trigger port may synchronize with the other ports of the connector. The possible interactions are the non empty sublists containing this trigger port.

In BIP, it is possible to associate with an interaction an activation condition (guard) and a data transfer function both written in C. The interaction is possible if components are ready to communicate through its ports and its activation condition is true. Its execution starts with the computation of data transfer function followed by notification of its completion to the interacting components.

4 Automatic Model Transformation from AADL to BIP

In this section, we present the translation from AADL [5] to BIP [9]. It is organized in five part. First, we translate AADL software components (subprogram, data, thread and process). Second, we translate hardware components (processor, device). Third, we translate a system component. Fourth, we translate the AADL annex behavior specification [1] in BIP. Finally, we translate connections.

4.1 Software Component

We define the translation of the different AADL software components into BIP.

Subprogram. Depending on its structure, we translate the AADL subprograms into atomic or compound BIP components:

As atomic BIP component : When the AADL subprogram does not contain subprogram calls and connections, it is modelled as an atomic component in BIP. Figure 4 shows such a component. This component has two ports *call* and *return*, because subprogram can be called from another subprogram or thread. It also has a particular state *IDLE* and two transitions to express the call and return to the *IDLE* state. The behavior is obtained from the annex as described in section 4.4.

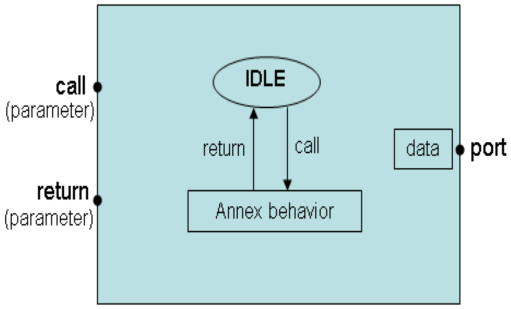


Fig. 4. Subprogram as atomic BIP component

As compound component : When the AADL subprogram contains subprogram calls and connections, it is modelled as a compound BIP component. The subprogram calls are executed sequentially. This execution is modelled by an atomic component with states $wait\_call_1 ... wait\_call_n$ and $wait\_return_1 ... wait\_return_n$, transitions labeled by the ports $call_1 ... call_n$ and $return_1 ... return_n$ (where n is the number of the subprograms called $sub_1 ... sub_n$). To enforce the right sequence of execution and the transfer of parameters, two ports *call* and *return* are used to express calls to the compound subprogram by other subprograms or threads, and the *port data* to sends event or data to the threads, as shown in Figure 5.

Data The data component type represents a data type in the source text that defines a representation and interpretation for instances of data in the source text. In BIP it is transformed into a C data structure.

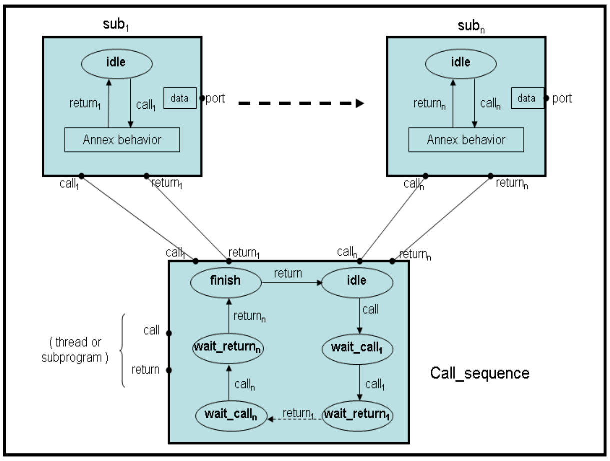


Fig. 5. Subprogram as compound BIP component

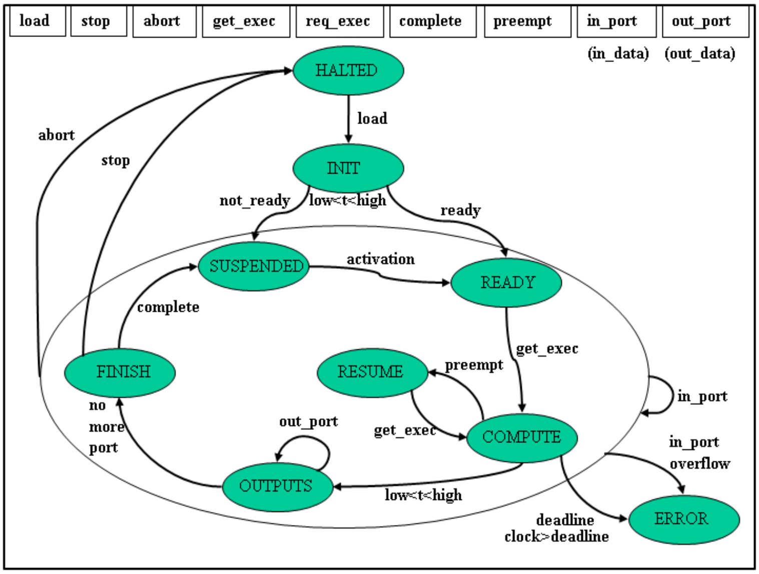


Fig. 6. BIP model for thread behavior

Thread : An AADL thread is modelled in BIP by an atomic component as shown in Figure 6. The initial state of the thread is HALTED. On an interaction through port *load* the thread is initialized. Once initialization is completed the thread enters the READY state, if the thread is ready for an interaction through the port *req_exec*. Otherwise, it enters the SUSPENDED state. When the thread is in the SUSPENDED state it cannot be dispatched for execution.

When in the SUSPENDED state, the thread is waiting for an event and/or period to be activated depending on the thread dispatch protocol (periodic, aperiodic, sporadic). In the READY state, a thread is waiting to be dispatched through an interaction in the port *get_exec*. When dispatched, it enters the state COMPUTE to make a computation. Upon successful completion of the computation, the thread goes to the OUTPUTS state. If there are some *out_ports* to dispatch the thread returns to the OUTPUTS state. otherwise, it enters the FINISH state.

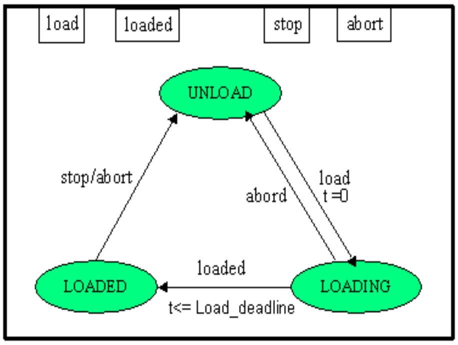


Fig. 7. BIP model for process behavior

The thread may be requested to enter its HALTED state through a port *stop* after completing the execution of a dispatch. A thread may also enter the thread HALTED state immediately through an *abort* port.

Process : Processes must contain at least one explicitly declared thread or thread group. The process behavior is illustrated in Figure 7. Once processors of an execution platform are started, the process enters to the state LOADING through port *load* and it is ready to be loaded.

A process is considered as stopped when all threads of the process are halted. When a process is stopped, each of its threads is given a chance to finalize its execution.

A process can be aborted by using *abort* port. In this case, all contained threads terminate their execution immediately and release all resources.

The *Load_deadline* property specifies the maximum amount of elapsed time allowed between the time the process begins and completes loading.

4.2 Execution Platform Components

This section defines the translation into BIP of processors and devices.

Processors. AADL processor components are an abstraction of hardware and software that is responsible for scheduling and executing threads. Schedulers are modelled as atomic BIP components as shown in Figure 8. The initial state of a scheduler is IDLE. When a thread become ready, the scheduler enters the CHOICE state through an interaction on port *ready*. In this state, the thread ID is stored into the scheduler memory. When a thread is dispatched, the scheduler selects a thread identifier (into *SelectedID* variable) and enters the WAIT_END state through an interaction on port *dispatch*. If there are several threads to be dispatched the scheduler re-enters to the state CHOICE, otherwise, it enters the state IDLE.

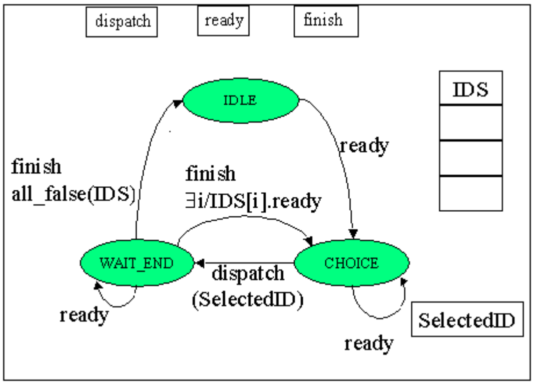


Fig. 8. BIP model of a scheduler

Devices. A device component represents an execution platform component that interfaces with the external environment. A device can interact with application software components through their ports. It is modelled as an atomic component in BIP.

4.3 System

A system component represents an assembly of software and execution platform components. All subcomponents of a system are considered to be contained in that system. A system is modelled as a compound component in BIP. Figure 9 shows a BIP component representing a system and connexion between threads, process, and scheduler.

4.4 Annex Behavior Specification

Some annex behavior elements can be directly translated to BIP whereas for others we need new BIP facilities. Actual behaviors are supposed to be described using the implementation language. The proposed behavioral annex allows the expression of data dependent behaviors so that more precise behavioral analysis remains possible.

- The *state variables* section declares typed identifiers. In BIP, they correspond to data variables. They must be initialized in the *initialization* section, which is directly included in the BIP initialization part.
- The *states* section declares automaton states as: The *initial* state is directly included in BIP. The *return* state indicates the return to the caller. This case is represented in BIP as a transition from *return* state to *idle* state.

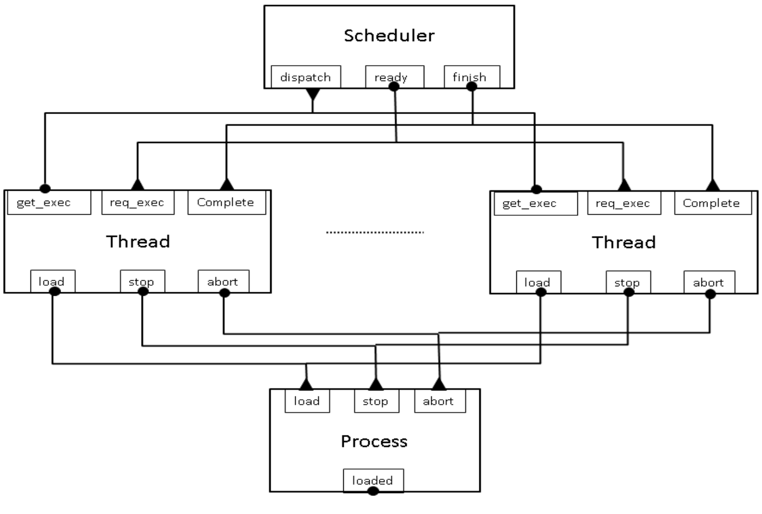


Fig. 9. BIP System

– The *transitions* section defines transitions from a source state to a destination state. Transitions can be guarded with events or boolean conditions, and can contain an action. Each transition is translated as one or a sequence of BIP transitions.

4.5 Connections

Port connection : is translated in BIP depending on the categories :

– an event connection is translated into strong synchronization between the corresponding event ports.
– a data connection is translated into connection with transfer of data.
– an event data connection is translated into a strong synchronization between the corresponding ports with transfer of data.

Parameter connection : is translated in BIP by transfer of data between the parameters of a sequence of subprogram calls in a thread, as shown in section 4.1.

5 Tool

From the high-integrity systems point-of-view, the use of automatic code generation in the development process is profitable. As the generated code is a combination of a relatively small set of extensively tested design patterns, the analysis and review of this code is easier than for hand-written code.

The tool chain is described in Figure 10, and it has the following features:

– *AADL to BIP Transformation:* Using model transformations, allows to perform analysis on the models before code generation. The tool generating BIP from AADL (Figure 10) has been implemented in Java, as a set of plugins for the open source Eclipse platform. It takes as input an AADL model(.aaxl) conforming to the AADL metamodel and generates a BIP model conforming to the BIP metamodel. Models generated may be timed or untimed. Timed models can be transformed into untimed models in which time progress is represented by a tick port that exists in all timed components and a connector connecting all tick ports.
– *Code Generation:* Takes as input a BIP model and generate the C/C++ code to be executed by the Engine.
– *Exploration Engine:* The engine has a state space exploration mode, which under some restrictions on the data used, generates state graphs that can be analyzed by using finite state model-checking tools.
– *Simulation:* Monitors the state of atomic components and finds all the enabled interactions by evaluating the guards on the connectors. Then, between the enabled interactions, priority rules are used to eliminate the ones with low priority.
– *Verification:* Automatic verification is very useful for early error detection.

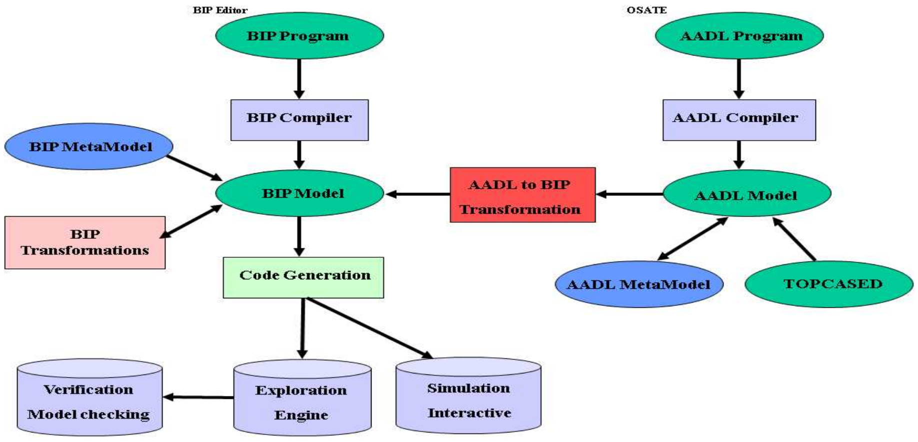


Fig. 10. AADL to BIP Tool Architecture

6 Case Studies

We used some examples of AADL [3,2] (with annex behavior specification) to check the feasibility of our translation from AADL to BIP. In this section, we present the example of a simplistic flight computer [2].

The Flight Computer has a thread called Sensor_Sim that periodically sends integers data for the current AoA(angle-of-attack) and Climb_Rate, and an event in case of Engine_Failure. It also has a thread called Stall_Monitor that is periodic and monitors the condition of the AoA and Climb_Rate sensors and raise a stall warning if certain conditions are met. The thread Operator simulates the pilot. It is a periodic thread that sends a command (Gear_Cmd) at every dispatch to raise or lower the landing gear of the aircraft. The thread Landing_Gear simulates the landing gear subsystem. It receives a command to start a landing gear operation, and is a sporadic thread with a minimum inter-arrival time of 3 seconds. The thread HCI is a human computer interface. It receives a Stall_Warning as an event data of type Integer; Engine_Failure as an event; a landing gear command from the pilot. It may send a landing gear operation request (Gear_Req) to the landing gear subsystem, and receives a landing gear operation acknowledgement (Gear_Ack) from the landing gear subsystem. It is a sporadic thread with a minimum inter-arrival time of 10ms. The graphical representation of Flight Computer system model is given in Figure 11.

6.1 BIP Model

The AADL model of the Flight Computer is transformed into BIP automatically by using our AADL to BIP translation tool. Figure 12 shows the obtained BIP model. This figure represents the BIP atomic components (AADL Threads) and

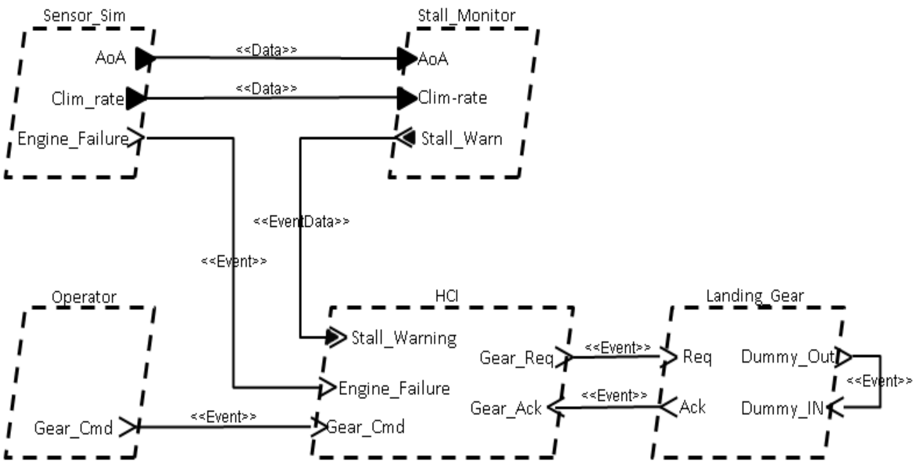


Fig. 11. Flight Computer Architecture

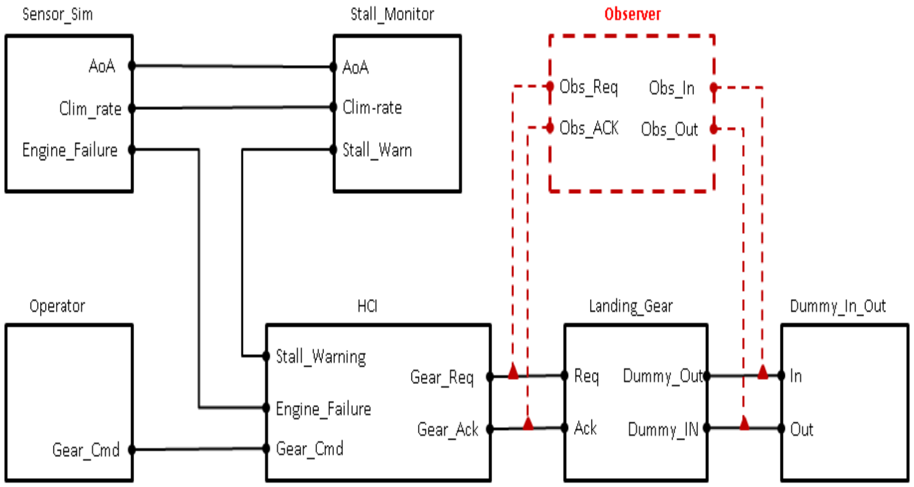


Fig. 12. BIP model for the Flight computer (including observer property, dashed)

connectors between them. Notice that we omit here the connectors between threads, process and scheduler that are shown in the Figure 9.

The component Dummy_In_Out models the communication between the Dummy _Out and Dummy_In events ports. In the AADL model (Figure 11), these two events are used to control thread reactivation: execution of the Landing_Gear thread is activated by the Dummy_In event; it emits a Dummy_Out event upon completion. Thus, synchronizing these two events ensures periodic activation of this thread.

6.2 Verification

The model construction methodology applied to this example, opens the way for enhanced analysis and early error detection by using verifications techniques.

Once the model has been generated, two model checking techniques for verification have been applied:

Model checking by Aldebaran: The first technique of verification is deadlock detection by using the tool Aldebaran [12]. Exhaustive exploration by the BIP exploration engine, generates a Labeled Transition System (LTS) which can be analyzed by model checking. e.g, Aldebaran takes as input the LTS generated from BIP and checks for deadlock-freedom. We have checked that the model is deadlock-free.

Model checking with observers: The second technique of verification is by using BIP observers to express and check requirements. Observers allow us to express in a much simple manner most safety requirements. We apply this technique to verify two properties:

- Verification of thread deadlines by using an observer component keeping track of the execution time of threads. If the execution time of a thread exceeds its deadline the observer moves to an error state.
- Verification of synchronization between components: Landing_Gear is sporadically activated bye HCI trough the Req port. When it is activated, it send back an acknowledgement through the ACK port, and possibly reactivates itself through the Dummy_In_Out component. This property can be verified by an observer which monitors the interactions between HCI, landing_Gear and Dummy_In_Out components (Figure 12).

7 Conclusion

The Architecture Analysis and Design Language (AADL) suffers from the absence of concrete operational semantics. In this paper, we address this problem by providing a translation from AADL to BIP, which has an operational semantics formally defined in terms of labelled transition systems. This translation allows simulation of AADL models, as well as application verification techniques, such as state exploration (using IF toolset [13]) or component-based deadlock detection (using Aldebaran [12], and D-Finder tool [11]). The proposed method has been implemented in translation tool, which has been tested on the Flight Computer case study, also presented in this paper. Future work includes incorporating features that will appear in V2.0 of the AADL standard.

References

1. Annex Behavior Specification SAE AS5506
2. `http://aadl.enst.fr/arc/doc/`

3. http://gforge.enseeiht.fr/docman/?group_id=37
4. http://www-verimag.imag.fr/~async/bip{M}etamodel.php
5. SAE. Architecture Analysis & Design Language (standard SAE AS5506) (September 2004), http://www.sae.org
6. SEI. Open Source AADL Tool Environment, http://la.sei.cmu.edu/aadlinfosite/OpenSourceAADLToolEnvironment.html
7. TOPCASED, http://www.topcased.org/
8. Basu, A., Bensalem, S., Gallien, M., Ingrand, F., Lesire, C., Nguyen, T.H., Sifakis, J.: Incremental component-based construction and verification of a robotic system. In: Proceedings of ECAI 2008, Patras, Greece (2008)
9. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in bip. In: Proceedings of SEFM 2006, Pune, India, pp. 3–12. IEEE Computer Society Press, Los Alamitos (2006)
10. Basu, A., Mounier, L., Poulhiès, M., Pulou, J., Sifakis, J.: Using bip for modeling and verification of networked systems – a case study on tinyos-based networks. In: Proceedings of NCA 2007, Cambridge, MA, USA, pp. 257–260 (2007)
11. Bensalem, S., Bozga, M., Sifakis, J., Nguyen, T.H.: Compositional verification for component-based systems and application. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311. Springer, Heidelberg (2008)
12. Bozga, M., Fernandez, J.-C., Kerbrat, A., Mounier, L.: Protocol verification with the aldebaran toolset. In: STTT, vol. 1, pp. 166–183 (1997)
13. Bozga, M., Graf, S., Ober, I., Ober, I., Sifakis, J.: The IF toolset. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 237–267. Springer, Heidelberg (2004)
14. Sifakis, J., Gossler, G.: Composition for component-based modeling. Science of Computer Programming 55, 161–183 (2005)
15. Poulhiès, M., Pulou, J., Rippert, C., Sifakis, J.: A methodology and supporting tools for the development of component-based embedded systems. In: Kordon, F., Sokolsky, O. (eds.) Monterey Workshop 2006. LNCS, vol. 4888, pp. 75–96. Springer, Heidelberg (2007)

Towards Model-Based Integration of Tools and Techniques for Embedded Control System Design, Verification, and Implementation

Joseph Porter, Gábor Karsai, Péter Völgyesi, Harmon Nine, Peter Humke, Graham Hemingway, Ryan Thibodeaux, and János Sztipanovits

Institute for Software Integrated Systems,
Vanderbilt University,
Nashville TN 37203, USA
jporter@isis.vanderbilt.edu,
http://www.isis.vanderbilt.edu

Abstract. While design automation for hardware systems is quite advanced, this is not the case for practical embedded systems. The current state-of-the-art is to use a software modeling environment and integrated development environment for code development and debugging, but these rarely include the sort of automatic synthesis and verification capabilities available in the VLSI domain. We present a model-based integration environment which uses a graphical architecture description language (EsMoL) to pull together control design, code and configuration generation, platform-specific simulation, and a number of other features useful for taming the heterogeneity inherent in safety-critical embedded control system designs. We describe concepts, elements, and development status for this suite of tools.

1 Introduction

Embedded software often operates in environments critical to human life and subject to our direct expectations. We assume that a handheld MP3 player will perform reliably, or that the unseen aircraft control system aboard our flight will function safely and correctly. Safety-critical embedded environments require far more care than provided by the current best practices in software development. Embedded systems design challenges are well-documented [1], but industrial practice still falls short of expectations for many kinds of embedded systems.

In modern designs, graphical modeling and simulation tools (e.g. Mathworks' Simulink/Stateflow) represent physical systems and engineering designs using block diagram notations. Design work revolves around simulation and test cases, with code generated from "'complete"' designs. Control designs often ignore software design constraints and issues arising from embedded platform choices. At early stages of the design, platforms may be vaguely specified to engineers as sets of tradeoffs [2].

Software development uses UML (or similar) tools to capture concepts such as components, interactions, timing, fault handling, and deployment. Workflows focus on source code organization and management, followed by testing and debugging on target hardware. Physical and environmental constraints are not represented by the tools. At best such constraints may be provided as documentation to developers.

Complete systems rely on both aspects of a design. Designers lack tools to model the interactions between the hardware, software, and the environment with the required fidelity. For example, software generated from a carefully simulated functional dataflow model may fail to perform correctly when its functions are distributed over a shared network of processing nodes. Cost considerations may force the selection of platform hardware that limits timing accuracy. Neither aspect of development supports comprehensive validation of certification requirements to meet government safety standards.

We propose a suite of tools that aim to address many of these challenges. Currently under development at Vanderbilt's Institute for Software Integrated Systems (ISIS), these tools use the Embedded Systems Modeling Language (ESMoL), which is a suite of domain-specific modeling languages (DSML) to integrate the disparate aspects of a safety-critical embedded systems design and maintain proper separation of concerns between engineering and software development teams. Many of the concepts and features presented here also exist separately in other tools. We describe a model-based approach to building a unified model-based design and integration tool suite which has the potential to go far beyond the state of the art.

We will provide an overview of the tool vision and describe features of these tools from the point of view of available functionality. Note that two development processes will be discussed – the development of a distributed control system implementation (by an assumed tool user), and our development of the tool suite itself. The initial vision section illustrates how the tools would be used to model and develop a control system. The final sections describe different parts of our tool-development process in decreasing order of maturity.

2 Toolchain Vision and Overview

In this work, we envision a sophisticated, end-to-end toolchain that supports not only construction but also the verification of the engineering artifacts (including software) for high-confidence applications. The development flow provided by the toolchain shall follow a variation of the classical V-model (with software and hardware development on the two branches), with some refinements added at the various stages. Fig. 1 illustrates this development flow.

Consider the general class of control system designs for use in a flight control system. Sensors, actuators, and data networks are designed redundantly to mitigate faults. The underlying platform implements a variant of the time-triggered architecture (TTA) [3], which provides precise timing and reliability guarantees. Safety-critical tasks and messages execute according to strict precomputed schedules to ensure synchronization between replicated components and provide fault mitigation and management. Software implementations of the control

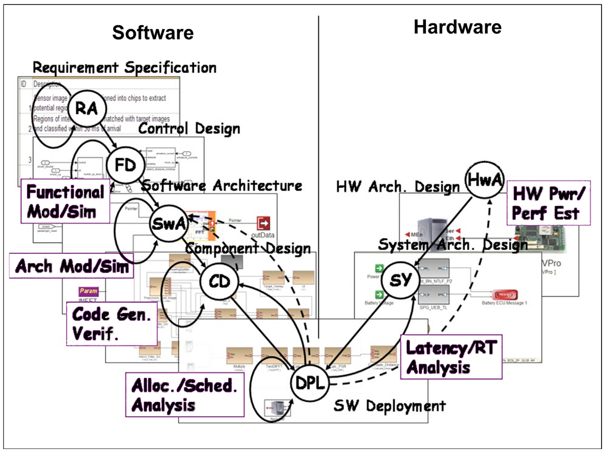


Fig. 1. Conceptual model of the toolchain: Development flow

functions must pass strict certification requirements which impose constraints on the software as well as on the development process.

A modeling language to support this development flow must have several desired properties: (1) the ability to capture the relevant aspects of the system architecture and hardware, (2) ability to "understand" (and import) functional models from existing design tools, (3) support for componentization of functional models, and (4) ability to model the deployment of the software architecture onto the hardware architecture. The ability to import existing models from functional modeling tools is not a deeply justified requirement, it is merely pragmatic. EsMoL provides modeling concepts and capabilities that are highly compatible with AADL [4]. The chief differences are that EsMoL aims for a simpler graphical entry language, a wider range of execution semantics, and most important model-enabled integration to external tools as described below. Model exchange with AADL tools may be desirable in the future. A simple sample design will introduce key points of our model-based development flow and illustrate language concepts.

Our language design was influenced by two factors: (1) the MoC implemented by the platform and (2) the need for integration with legacy modeling and embedded systems tools. We have chosen Simulink/Stateflow as the supported "legacy" tool. As our chosen MoC relies on periodically scheduled time-triggered components, it was natural to use this concept as the basis for our modeling language and interpret the imported Simulink blocks as the implementation of these components. To clarify the use of this functionality, we import a Simulink design and select functional subsets which execute in discrete time, and then assign them to software components using a modeling language that has compatible (time-triggered) semantics. Communication links (signals) between Simulink blocks are mapped onto TTA messages passed between the tasks. The resulting language provides a componentized view of Simulink models that are scheduled periodically (with a fixed rate) and communicate using scheduled, time-triggered messages. Extensions to heterogeneous MoC-s is an active area of research.

2.1 Requirements Analysis (RA)

Our running example will model a data network implementing a single sensor/actuator loop with a distributed implementation. The sensors and actuators in the example are doubly-redundant, while the data network is triply-redundant. The common nomenclature for this type of design is TMR (triple modular redundancy). Unlike true safety-critical designs, we will deploy the same functions on all replicas rather than requiring multiple versions as is often done in practice [5]. The sensors and actuators close a single physical feedback loop. Specifying the physical system and particulars of the control functions are beyond the scope of this example as our focus is on modeling.

This example has an informal set of requirements, though our modeling language currently supports the formalization of timing constraints between sensor and actuator tasks. Formal requirements modeling offers great promise, but in ESMoL requirements modeling is still in conceptual stages. A simple sensor/actuator latency modeling example appears in a later section covering preliminary features for the language.

2.2 Functional Design (FD)

Functional designs can appear in the form of Simulink/Stateflow models or as existing C code snippets. ESMoL does not support the full semantics of Simulink. In ESMoL the execution of Simulink data flow blocks is restricted to periodic discrete time, consistent with the underlying time-triggered platform. This also restricts the type and configuration of blocks that may be used in a design. Continuous integrator blocks and sample time settings do not have meaning in ESMoL. C code snippets are allowed in ESMoL as well. C code definitions are limited to synchronous, bounded response time function calls which will execute in a periodic task.

Fig. 2 shows a simple top-level Simulink design for our feedback loop along with the imported ESMoL model (Fig. 3). The ESMoL model is a structural replica of the original Simulink, only endowed with a richer software design

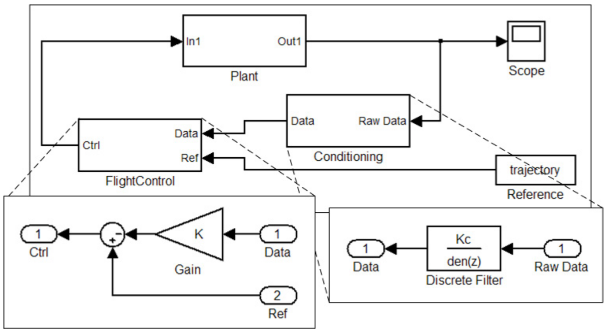


Fig. 2. Simulink design of a basic signal conditioner and controller

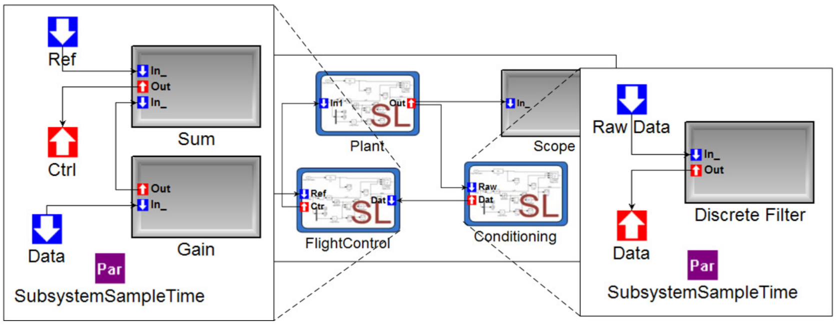


Fig. 3. ESMoL-imported functional models of the Simulink design

environment and tool-provided APIs for navigating and manipulating the model structure in code. A model import utility provides the illustrated function.

2.3 Software Architecture (SwA)

The software architecture model describes the logical interconnection of functional blocks. In the architecture language a component may be implemented by either a Simulink Subsystem or a C function. They are compatible at this level, because here their model elements represent the code that will finally implement the functions. These units are modeled as blocks with ports, where the ports represent parameters passed into and out of C function calls. Semantics for SwA Connections are those of task-local synchronous function invocations as well as message transfers between tasks using time-triggered communication.

Fig. 4 shows the architecture diagram for our TMR model. Instances of the functional blocks from the Simulink model are augmented with C code implementing replicated data voting.

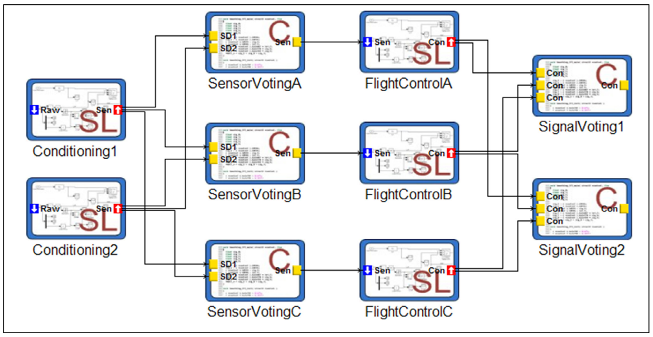


Fig. 4. The architecture diagram defines logical interconnections, and gives finer control over instantiation of functional units

2.4 Hardware Architecture (HwA)

Hardware configurations are explicitly modeled in the platform language. Platforms are defined hierarchically as hardware units with ports for interconnections. Primitive components include processing nodes and communication buses. Behavioral semantics for these networks come from the underlying time-triggered architecture. The platform provides services such as deterministic execution of replicated components and timed message-passing. Model attributes for hardware also capture timing resolution, overhead parameters for data transfers, and task context switching times.

Figs. 5 and 6 show model details for redundant hardware elements. Each controller unit is a private network with two nodes and three independent data buses. Sensor voting and flight control instances are deployed to the controller unit networks.

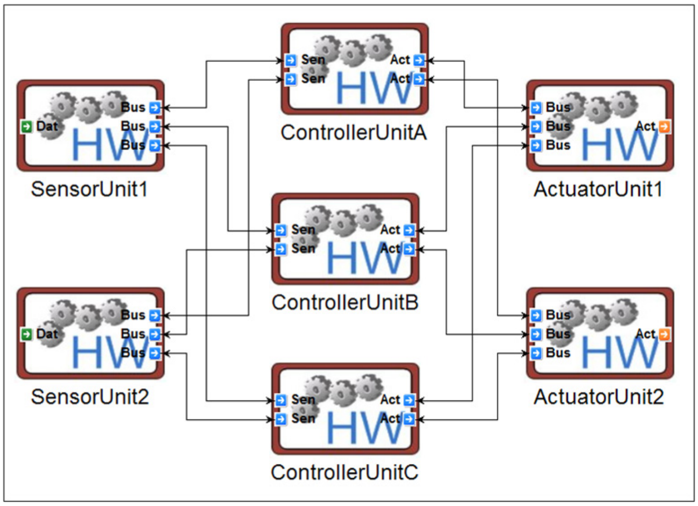


Fig. 5. Overall hardware layout for the TMR example

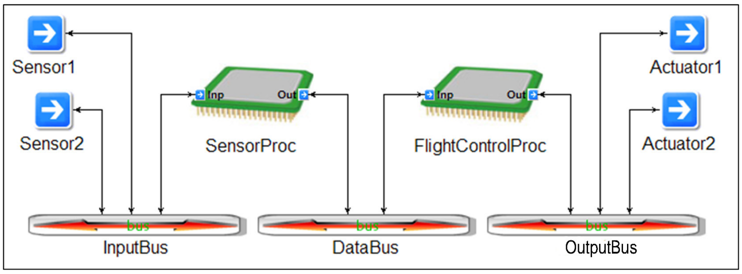


Fig. 6. Detail of hardware model for controller units

2.5 Deployment Models (CD, SY, DPL)

A common graphical language captures the grouping of architecture components into tasks. This language represents three of the design stages from the V-diagram (Fig. 1) – component design (CD), system architecture design (SY), and software deployment (DPL), though we will refer to it as the deployment language. In ESMoL a task executes on a processing node at a single periodic rate. All components within the task execute synchronously. Data sent between tasks takes the form of messages in the model. Whether delivered locally (same processing node) or remotely, all inter-task messages are pre-scheduled for delivery. ESMoL uses logical execution time semantics found in time-triggered languages such as Giotto [6] – message delivery is scheduled after the deadline of the sending task, but before the release of the receiving tasks. In the TT model message receivers assume that required data is already available at task release time. Tasks never block, but execute with whatever data is available each period.

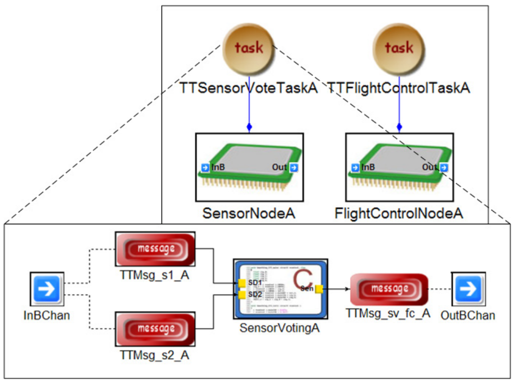


Fig. 7. Deployment model: task assignment to nodes and details of task definition

Deployment concepts – tasks running on processing nodes and messages sent over data buses – are modeled as shown in Fig. 7. Software components and bus channels are actually references to elements defined in architecture and platform models. Model interpreters use deployment models to generate platform-specific task wrapping and communication code as well as analysis artifacts.

3 Existing Tools: Simulink to TTA

Control designs in Simulink are integrated using a graphical modeling language describing software architecture. Components within the architecture are assigned to tasks, which run on nodes in the platform.

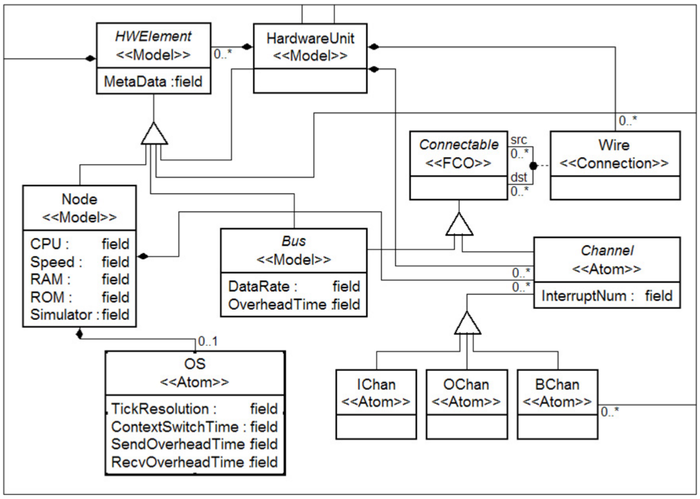


Fig. 8. Platforms. This metamodel describes a simple language for modeling the topology of a time-triggered processing network. A sample platform model is included.

3.1 Integration Details

The Simulink and Stateflow sublanguages of our modeling environment are described elsewhere, though the ESMoL language changes many of the other design concepts from previously developed languages described by Neema [7].

In our toolchain we created a number of code generators. To construct the two main platform-independent code generators (one for Simulink-style models and another one for Stateflow-style models) we have used a higher-level approach based on graph transformations [8]. This approach relies on assumptions that (1) models are typed and attributed graphs with specific structure (governed by the metamodel of the language) and (2) executable code can be produced as an abstract syntax graph (which is then printed directly into source code). This transformation-based approach allows a higher-level representation of the translation process, which lends itself more easily to automatic verification.

The models in the example and the metamodels described below were created using the ISIS Generic Modeling Environment tool (GME) [9]. GME allows language designers to create stereotyped UML-style class diagrams defining metamodels. The metamodels are instantiated into a graphical language, and metamodel class stereotypes and attributes determine how the elements are presented and used by modelers. The GME metamodeling syntax may not be entirely familiar to the reader, but it is well-documented in Karsai et al [10]. Class concepts such as inheritance can be read analogously to UML. Class aggregation represents containment in the modeling environment, though an aggregate element can be flagged as a port object. In the modeling environment a port object will also be visible at the next higher level in the model hierarchy, and available for connections. The dot between the Connectable class and the Wire class represents a line-style connector in the modeling environment.

High-confidence systems require platforms providing services and guarantees for properties such as fault containment, temporal firewalls, partitioning, etc. System developers should not re-implement such critical services from scratch [2]. Note that the platform also defines a 'Model of Computation' [11]. An MoC governs how the concurrent objects of an application interact (i.e. synchronization and communication), and how these activities unfold in time. The simple platform definition language shown in Fig. 8 contains relationships and attributes describing time-triggered networks.

Similarly, Fig. 9 shows the software architecture language. Connector elements model communication between components. Semantic details of such interactions remain abstract in this logical architecture – platform models must be defined

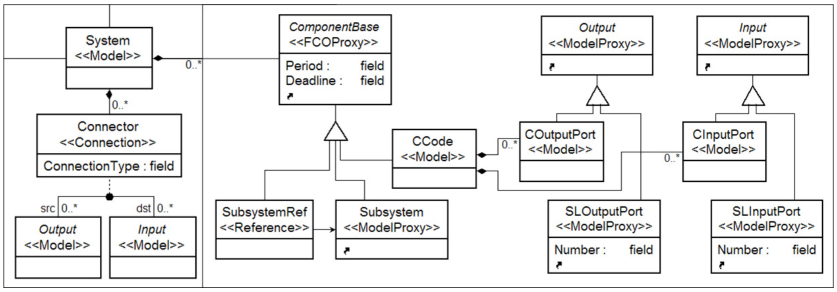


Fig. 9. Architecture Metamodel. Architecture models use Simulink subsystems or C code functions as components, adding attributes for real-time execution. The Input and Output port classes are typed according to the implementation class to which they belong.

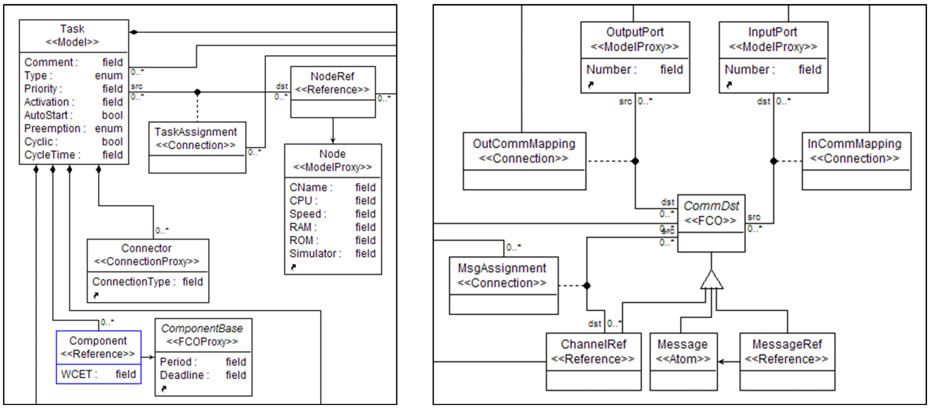


Fig. 10. Details from deployment sublanguage

and associated in order to completely specify interactions (though this version only offers synchronous or time-triggered communications).

Deployment models capture the assignment of Components (and Ports) from the Architecture to Platform Nodes (and Channels). Additional implementation details (e.g. worst-case execution time) are represented here for platform-specific synthesis. Fig. 10 shows the relevant modeling concepts. Simulink objects SLInputPort and SLOutputPort are assigned to Message objects, which represent the marshaling of data to be sent on a Bus.

4 Under Development: Platform-Specific Simulation, Generic Hardware, and Scheduling

A control system designer initially uses simulation to check correctness of the design. Software engineers later take code implementing control functions and deploy it to distributed controllers. Concurrent execution and platform limitations may introduce new behaviors which degrade controller performance and introduce errors. Ideally, the tools could allow the control functions to be resimulated with appropriate platform effects.

The TrueTime simulation environment [12] provides Simulink blocks modeling processing nodes and communication links. ESMoL tasks map directly to TrueTime tasks. In TrueTime, tasks can execute existing C code or invoke subsystems in Simulink models. Task execution follows configured real-time scheduling models, with communication over a selected medium and protocol. TrueTime models use a Matlab script to associate platform elements with function implementations. A platform-specific re-simulation requires this Matlab mapping function, and in our case also a periodic schedule for distributed time-triggered execution. Both of these can be obtained by synthesis from ESMoL models.

Resimulation precedes synthesis to a time-triggered platform. In order to use generic computing hardware with this modeling environment, we created a simple, open, portable time-triggered virtual machine (VM) to simulate the timed behavior of a TT cluster [13] on generic processing hardware. Since the commercial TT cluster and the open TT VM both implement the same model of computation, synthesis differences amount to management of structural details in the models. The open VM platform is limited to the timing precision of the underlying processor, operating system, and network, but it is useful for testing.

For both steps above the missing link is schedule generation. In commercial TTP platforms, associated software tools perform cluster analysis and schedule generation. For resimulation and deployment to an open platform, an open schedule generation tool is required. To this end we created a schedule generator using the Gecode constraint programming library [14]. The scheduling approach implements and extends the work of Schild and Würtz [15]. Configuration for the schedule generator is also generated by the modeling tools.

4.1 Integration Details

To configure TrueTime or the scheduler, the important details lie in the deployment model. Tasks and Messages must be associated with the proper processing nodes and bus channels in the model. The ISIS UDM libraries [16] provide a portable C++ API for creating model interpreters, navigating in models, and extracting required information. See Fig. 10 for the relevant associations. Model navigation in these interpreters must maintain the relationships between processors and tasks and between buses and messages. Scheduler configuration also requires extraction of all message sender and receiver dependencies in the model.

5 Designs in Progress: Requirements and Model Updates

Many types of requirements apply to real-time embedded control systems design. Embedded systems are heterogeneous, so requirements can include constraints on control performance, computational resources, mechanical design, and reliability, to name a few things. Formal safety standards (e.g. DO-178B [5]) impose constraints on the designs as well as on the development process itself. Accordingly, current research has produced many techniques for formalizing requirements (e.g. ground models in abstract state machines [17] or Z notation [18]). Models could be used to incorporate formal requirements into other aspects of the design process. During analysis, requirements may appear as constraints in synthesized optimization problems or conditions for model checking. Requirements can also be used for test generation and assessment of results.

Management of model updates is also essential. As designs evolve engineers and developers reassess and make modifications. Changes to either the platform model or functional aspects of the design may invalidate architecture and deployment models created earlier. Some portions of the dependent models will survive changes. Other parts needing changes must be identified. Where possible, updates should be automated.

5.1 Integration Details

The requirements sublanguage is in design, and so is light on details. As a simple example of the potential of such a language, Fig. 13 shows a model with latency requirements between tasks, and Fig. 11 shows the modeling language definition. This simple relationship can be quantified and passed directly to the schedule solver as a constraint. Ideally a more sophisticated requirements language could capture the syntax and semantics of an existing formal requirements tool. Some candidate languages and approaches are currently under consideration for inclusion in the framework.

To track model changes we propose to use the Simulink UserData field to store a unique tag in each functional block when the models are imported. During an

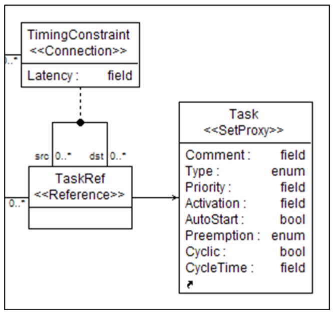


Fig. 11. Latencies are timing constraints between task execution times

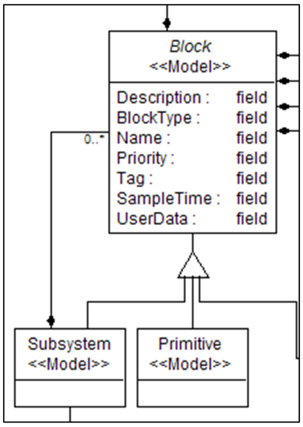
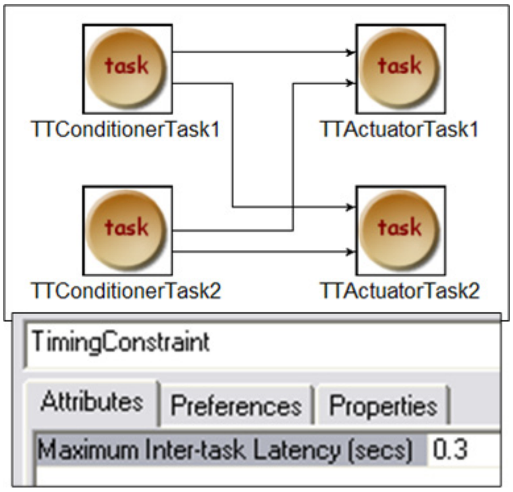


Fig. 12. Simulink's UserData field can help manage model changes occurring outside the design environment



Maximum Inter-task Latency (secs) 0.3

Fig. 13. Example of task latency spec for sample model, with detail of timing attribute value specified on model links

update operation tags in the control design can be compared with previously imported tags in the model environment. Fig. 12 shows the UserData attribute from our Simulink sublanguage, corresponding to the actual attribute in Simulink blocks. To handle issues arising from topology concerns during model evolution, we require control designers to group top-level functionality into subsystems and place a few restrictions on model hierarchy in deployment models.

6 Wishlist: Expanded Semantics, Implementation Generation, and Verification

Many exciting possibilities loom on the horizon for this tool chain construction effort. We briefly describe some concepts currently in discussion for the tools.

The current modeling languages describe systems which provide performance and reliability guarantees by implementing a time-triggered model of computation. This is not adequate for many physical processes and controller platforms. We also need provisions for event-triggered communication and components. Event-triggered component structures give rise to interesting and useful communication patterns common in practical systems (e.g. publish-subscribe, rendezvous, and broadcast). Several research projects have explored heterogeneous timed models of computation. Two notable examples are the Ptolemy project [19] and the DEVS formalism and associated implementations [20]. More general simulation and model-checking tools for timed systems and specifications include UPPAAL [21] and timed abstract state machines [22]. We aim to identify useful design idioms from event-triggered models and extend the semantics of the modeling language to incorporate them. Synthesis to analysis tools is also possible using model APIs.

Safe automation of controller implementation techniques is another focus. Control designs are often created and simulated in continuous time and arbitrary numerical precision, and then discretized in time for platforms with periodic sampling and in value for platforms with limited numeric precision. Recent work in optimization and control offers some techniques for building optimization problems which describe valid controller implementation possibilities [23] [24]. Early work on model interpreters aims to generate such optimization problems directly from the models. Other interesting problems include automated generation of fixed-point scaling for data flow designs. If integrated, tools like BIP [25] provide potential for automated verification of distributed computing properties (safety, liveness, etc...). Model representation of data flow functions, platform precision, and safety requirements could be used together for scaling calculation.

The addition of proper formal requirements modeling can enable synthesis of conditions for model checking and other verification tools. Executable semantics for these modeling languages can also provide the behavioral models to be checked (see Chen [26] [27], Gargantini [28], and Ouimet [29]). Other relevant work includes integration of code-level checking, as in the Java Pathfinder [30] or Saturn [31] tools. Synthesis to these tools must also be verified, an active area of research at ISIS [32].

Acknowledgements

This work is sponsored in part by the National Science Foundation (grant NSF-CCF-0820088) and by the Air Force Office of Scientific Research, USAF (grant/contract number FA9550-06-0312). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

References

1. Henzinger, T., Sifakis, J.: The embedded systems design challenge. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 1–15. Springer, Heidelberg (2006)
2. Sangiovanni-Vincentelli, A.: Defining Platform-based Design. EEDesign of EE-Times (February 2002)
3. Kopetz, H., Bauer, G.: The time-triggered architecture. In: Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software (October 2001)
4. AS-2 Embedded Computing Systems Committee: Architecture analysis and design language (AADL). Technical Report AS5506, Society of Automotive Engineers (November 2004)
5. RTCA, Inc. 1828 L St. NW, Ste. 805, Washington, D.C. 20036: DO-178B: Software Considerations in Airborne Systems and Equipment Certification. Prepared by: RTCA SC-167 (December 1992)
6. Henzinger, T.A., Horowitz, B., Kirsch, C.M.: Giotto: A time-triggered language for embedded programming. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 166–184. Springer, Heidelberg (2001)
7. Neema, S., Karsai, G.: Embedded control systems language for distributed processing (ECSL-DP). Technical Report ISIS-04-505, Institute for Software Integrated Systems, Vanderbilt University (2004)
8. Agrawal, A., Karsai, G., Neema, S., Shi, F., Vizhanyo, A.: The design of a language for model transformations. Journal on Software and System Modeling 5(3), 261–288 (2006)
9. ISIS, V.U.: Generic Modeling Environment, `http://repo.isis.vanderbilt.edu/`
10. Karsai, G., Sztipanovits, J., Ledeczi, A., Bapty, T.: Model-integrated development of embedded software. Proceedings of the IEEE 91(1) (2003)
11. Lee, E.A., Sangiovanni-Vincentelli, A.L.: A denotational framework for comparing models of computation. Technical Report UCB/ERL M97/11, EECS Department, University of California, Berkeley (1997)
12. Ohlin, M., Henriksson, D., Cervin, A.: TrueTime 1.5 Reference Manual. Dept. of Automatic Control, Lund University, Sweden (January 2007), `http://www.control.lth.se/truetime/`
13. Thibodeaux, R.: The specification and implementation of a model of computation. Master's thesis, Vanderbilt University (May 2008)
14. Schulte, C., Lagerkvist, M., Tack, G.: Gecode: Generic Constraint Development Environment, `http://www.gecode.org/`
15. Schild, K., Würtz, J.: Scheduling of time-triggered real-time systems. Constraints 5(4), 335–357 (2000)
16. Magyari, E., Bakay, A., Lang, A., et al.: Udm: An infrastructure for implementing domain-specific modeling languages. In: The 3rd OOPSLA Workshop on Domain-Specific Modeling (October 2003)
17. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, Heidelberg (2003)
18. ISO/IEC: Information Technology – Z Formal Specification Notation – Syntax, Type System and Semantics. 13568:2002 (July 2002)

19. UCB: Ptolemy II, `http://ptolemy.berkeley.edu/ptolemyII/`
20. Hwang, M.H.: DEVS++: C++ Open Source Library of DEVS Formalism (May 2007), `http://odevspp.sourceforge.net/`
21. Basic Research in Computer Science (Aalborg Univ.) Dept. of Information Technology (Uppsala Univ.): Uppaal. Integrated tool environment for modeling, validation and verification of real-time systems, `http://www.uppaal.com/`
22. Ouimet, M., Lundqvist, K.: The timed abstract state machine language: An executable specification language for reactive real-time systems. In: Proceedings of the 15th International Conference on Real-Time and Network Systems (RTNS 2007), Nancy, France (March 2007)
23. Skaf, J., Boyd, S.: Controller coefficient truncation using lyapunov performance certificate. IEEE Transactions on Automatic Control (in review) (December 2006)
24. Bhave, A., Krogh, B.H.: Performance bounds on state-feedback controllers with network delay. In: IEEE Conference on Decision and Control 2008 (submitted) (December 2008)
25. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: SEFM 2006: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods, pp. 3–12. IEEE Computer Society Press, Washington (2006)
26. Chen, K., Sztipanovits, J., Abdelwahed, S.: A semantic unit for timed automata based modeling languages. In: Proceedings of RTAS 2006, pp. 347–360 (2006)
27. Chen, K., Sztipanovits, J., Abdelwalhed, S., Jackson, E.: Semantic anchoring with model transformations. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 115–129. Springer, Heidelberg (2005)
28. Gargantini, A., Riccobene, E., Rinzivillo, S.: Using spin to generate testsfrom ASM specifications. In: Börger, E., Gargantini, A., Riccobene, E. (eds.) ASM 2003. LNCS, vol. 2589, pp. 263–277. Springer, Heidelberg (2003)
29. Ouimet, M., Lundqvist, K.: Automated verification of completeness and consistency of abstract state machine specifications using a sat solver. In: 3rd International Workshop on Model-Based Testing (MBT 2007), Satellite of ETAPS 2007, Braga, Portugal (April 2007)
30. Visser, W., Havelund, K., Brat, G., Park, S., Lerda, F.: Model checking programs. Automated Software Engineering Journal 10(2) (April 2003)
31. Xie, Y., Aiken, A.: Saturn: A sat-based tool for bug detection. In: Proceedings of the 17th International Conference on Computer Aided Verification, pp. 139–143 (January 2005)
32. Narayanan, A., Karsai, G.: Towards verifying model transformations. In: Bruni, R., Varró, D. (eds.) 5th International Workshop on Graph Transformation and Visual Modeling Techniques, 2006, Vienna, Austria, pp. 185–194 (April 2006)

Challenges in Model-Driven Software Engineering

Ragnhild Van Der Straeten[1], Tom Mens[2], and Stefan Van Baelen[3]

[1] Software and Systems Engineering Lab, Vrije Universiteit Brussel
rvdstrae@vub.ac.be
[2] Service de Génie Logiciel, Université de Mons
tom.mens@umons.ac.be
[3] DistriNet, Department of Computer Science, K.U.Leuven
Stefan.VanBaelen@cs.kuleuven.be

Abstract. After more than a decade of research in Model-Driven Engineering (MDE), the state-of-the-art and the state-of-the-practice in MDE has significantly progressed. Therefore, during this workshop we raised the question of how to proceed next, and we identified a number of future challenges in the field of MDE. The objective of the workshop was to provide a forum for discussing the future of MDE research and practice. Seven presenters shared their vision on the future challenges in the field of MDE. Four breakout groups discussed scalability, consistency and co-evolution, formal foundations, and industrial adoption, respectively. These themes were identified as major categories of challenges by the participants. This report summarises the different presentations, the MDE challenges identified by the workshop participants, and the discussions of the breakout groups.

1 Introduction

MoDELS'08 is already the eleventh conference on UML, modelling and model-driven engineering (MDE). After more than a decade, research in MDE has significantly evolved and improved. Nevertheless, still a lot of fundamental and practical issues remain. A recent article by France and Rumpe [1] described some challenges to realise the MDE vision of software development. The existence of the article and a number of recent events show the real need for a forum to discuss future challenges in MDE. One such forum was the Dagstuhl Perspectives Workshop 08331 on *"Model Engineering of Complex Systems"*[1] which was organised at Dagstuhl, Germany on August 10-13, 2008.

The MoDELS workshop on *"Challenges in Model-Driven Software Engineering"* can be considered as a continuation of the discussions held during this Dagstuhl seminar. More specifically, our workshop addressed the question of how to proceed next, by identifying the future short-term and long-term challenges in MDE, and proposing ways to address these challenges. The main objective of the workshop was to provide a forum for people from academia and industry to:

[1] http://kathrin.dagstuhl.de/08331/

– identify obstacles to MDE research and practice;
– facilitate transfer of research ideas to industry;
– propose "revolutionary" novel ideas;
– proclaim important challenges that are either fundamental or pragmatic.

Our initiative was strengthened by the MoDELS'08 panel on the *"Past and Future of MDD"* that took place three days after our workshop. Partly based on questions raised during our workshop, panelists presented their vision of how MDD technologies and particular aspects of model-driven development research will evolve over the next 10 or more years.

This report summarises the presentations, the discussions held, and the challenges identified during the workshop. It can be used as a guide to researchers in software engineering to help them plan their future research, and to convince policy makers of the continuing and increasing importance of MDE research.

2 About the Workshop

The event was one of the most successful workshops co-located with MoDELS'08. There were 66 registered participants coming from 19 different countries, of which 5 non-European ones. From a gender perspective, about 21% of the participants were female.

In total, we received 15 submissions, of which 11 were accepted. Seven of those were invited to present their ideas during the workshop. A short abstract of each presentation is listed below. The articles of the three presentations marked with (*) are included in this Workshop Reader. For the other articles, as well as for the accepted submissions that were not presented during the workshop, please consult the electronic workshop proceedings [2].

Parastoo Mohagheghi (*). *MDE Adoption in Industry: Challenges and Success Criteria* [3]
MDE has been promoted to solve one of the main problems faced by software industry today: coping with the complexity of software development by raising the abstraction level and introducing more automation in the process. The promises are many. Among them are: improved software quality by increased traceability between artifacts, early defect detection, reducing manual and error-prone work and including knowledge in generators. However, to be successfully adopted by industry, MDE must be supported by a rich ecosystem of stable, compatible and standardised tools. It should also not introduce more complexity than it removes. The presentation reported on the authors' experience in adoption of MDE in industrial and research projects. It also discussed the areas in which MDE has potential for success and what the key success criteria are.

Dimitrios Kolovos (*) *Scalability: The Holy Grail of MDE* [4]
Scalability is a desirable property in MDE. The current focus of research in MDE is on declarative languages for model management, and scalable mechanisms for persisting models. The presenter claimed that, instead, modularity and encapsulation in modelling languages should be the main focus. This

claim was justified by demonstrating how those two principles apply to a related domain, namely code development, where the issue of scalability has been addressed to a much greater extent than in MDE.

Ernesto Posse *A foundation for MDE* [5]
MDE is still lacking adoption by developers. To live up to its full potential MDE must rest on a solid foundation. Therefore, one of the main challenges facing MDE today is the establishment of such a foundation. In this presentation, UML-RT was used as a case study to illustrate what can be achieved, what is missing and what kind of issues must be addressed by a successful approach to MDE.

Antonio Vallecillo (*) *Behaviour, Time and Viewpoint Consistency: Three Challenges for MDE* [6]
Three problems that MDE should tackle in order to be useful in industrial environments were outlined in this presentation. Firstly, the specification of the behavioural semantics of meta-models so that different kinds of analysis can be conducted, e.g., simulation, validation and model checking. A second challenge is the support of the notion of time in these behavioural descriptions, to be able to conduct, e.g., realistic performance and reliability analysis of industrial systems. As a third challenge, not only the accidental complexity involved in building software systems needs to be tackled, but their essential complexity should be addressed too. To achieve this, more effective use needs to be made of independent but complementary viewpoints to model large-scale systems, and correspondences between them to reason about the consistency of the global specifications need to be specified.

Dennis Wagelaar *Challenges in bootstrapping a model-driven way of software development* [7]
According to the presenter, current MDE technologies are often demonstrated using well-known scenarios that consider the MDE infrastructure to be already in place. If developers need to develop their own infrastructure because existing tools are insufficient, they will encounter a number of challenges. Generally, developers cannot just implement all their model transformations and other MDE infrastructure immediately, because it simply takes too long before they get usable results. An incremental approach to putting model-driven development into place gives you the necessary "breakpoints", but poses extra challenges with regard to the MDE technologies used. Some of these challenges are: how to bootstrap a step-wise refinement chain of model transformations, how to bootstrap the modelling language usage, how to fit in round-trip engineering, and what are the useful properties for a model transformation tool.

Robert Clarisó *UML/OCL Verification in Practice* [8]
One of the promises of model-driven development is the ability to identify defects early, at the level of models, which helps to reduce development costs and improve software quality. However, there is an emerging need for "lightweight" model verification techniques that are usable in practice, i.e., able to find and notify defects in realistic models without requiring a strong verification background or extensive model annotations. Some promising

approaches revolve around the satisfiability property of a model, i.e., deciding whether it is possible to create a well-formed instantiation of the model. Existing solutions in the UML/OCL context were discussed. The presenter claimed that this problem has not yet been satisfactorily addressed.

Jordi Cabot *Improving Requirements Specifications in Model-Driven Development Processes* [9]
Understanding the organisational context and rationales that lead up to system requirements helps to analyse the stakeholders' interests and how they might be addressed or compromised by the different design alternatives. These aspects are very important for the ongoing success of the system but are not considered by current MDE methods. The presenter argued for the necessity of extending existing methods with improved requirement techniques based on goal-oriented techniques for the analysis and specification of the organisation context, and discussed the benefits and challenges of such integration.

3 Identified Challenges

During the plenary session that took place after the presentations, all workshop participants identified some of the major challenges in MDE. Those challenges are enumerated below. It is important to note that this list is inevitably incomplete. Also, the order in which we present the challenges here is of no particular importance.

Model quality. We need to deal with quality aspects in modelling and model-driven engineering. This gives rise to a number of open questions:
- How can we define model quality?
- How can we assure, predict, measure, improve, control, manage quality?
- How can we reconcile conflicting quality aspects?

These and many related challenges are very important, and have been discussed in more detail in the MoDELS'08 workshop on *"Quality in Modelling"*. There is also a recent book that addresses these topics [10].

Run-time models. In model-driven software engineering focus has been primarily on using models at analysis, design, implementation, and deployment stages of development. The use of models during run-time extends the use of modelling techniques beyond the design and implementation phases of development and introduces a number of challenges:
- How can we represent dynamic behaviour?
- What should a run-time model look like? How can we use and maintain such models at run-time?
- How do they relate to "static" models?
- What are good approaches to follow when developing run-time models?
- What are the differences, advantages and shortcomings of model interpretation, model simulation/execution and code generation techniques?

These and many related challenges have been discussed during the MoDELS'08 workshop on *"Models@run.time"*.

Requirements modelling. Research related to requirements is underrepresented in the MDE community. Nevertheless a number of important challenges remain to be tackled:

- How can we model requirements?
- How can we bridge the gap between informal (textual) requirement specifications and formal requirement models?
- How can we integrate the activity of requirement specifications into traditional modelling?
- How can we achieve traceability between requirement specifications and design models?

Standards and benchmarks. There is a need for standards and benchmarks to compare different tools and approaches. Benchmarks provide an excellent resource to measure progress and the significance of a contribution. However, widely accepted benchmarks do not exist yet. This leads to the following open questions:

- How to design and develop benchmarks that facilitate comparison between tools and approaches?
- Which standards are needed to facilitate interoperability between tools?
- How can we obtain and share common data (models, model transformations, case studies)?

Modelling languages. Models cannot be developed without precise modelling languages. Modelling languages are one of the main themes of the MoDELS conferences. Although the state of research in modelling languages has significantly progressed, a number of open questions remain:

- Which languages, methods, principles and tools are necessary to design precise meta-models?
- How can we support better modularity in MDE?
- How to describe design *pragmatics* (as opposed to syntax and semantics)?
- How can we allow for and deal with multi-models and multi-formalism modelling?

Domain-specific modelling. Domain-specific modelling languages are designed to provide precise abstractions of domain-specific constructs. Models for complex systems encompass several domains. Capturing all important aspects of such a complex system requires developing multiple models using different DSMLs and introduces many challenges.

- How to develop and integrate models using different domain-specific modelling languages?
- What process and which tools should we use to analyse, design, develop, verify and validate domain-specific models and languages?
- How can we increase reuse across different domain-specific modelling languages?
- How can we facilitate code generation from domain-specific modelling languages?

- What is the trade-of between general-purpose modelling languages, tools, techniques and domain-specific ones? Do we get a higher degree of specialisation, higher expressiveness, and higher potential for code generation, model execution and formal reasoning?

Panelists of the MoDELS'08 panel on *"Addressing the Challenges of Multi-Modelling for Domain-Specific Modelling Languages"* have commented on these and related challenges.

Empirical analysis. In the context of MDE, the topic of empirical analysis raises a number of interesting challenges:

- What are the main obstacles and potential remedies when performing empirical studies of MDE?
- What are the strengths and weaknesses of evaluating MDE activities, tools and techniques in laboratory and field settings, as well as industrial case studies?
- How should we deal with the unavoidable trade-offs between realism and control?
- How can we obtain adequate estimations for an MDE process and which measurements are relevant for MDE?

These and related challenges have been addressed and discussed at the MoDELS'08 workshop on *"Empirical Studies in Model Driven Engineering"*.

Model verification and validation. As in any software development approach, verification and validation are essential for MDE. In the context of MDE, it imposes a number of additional challenges:

- How can we verify, validate, debug, and test the models and the code generated from those models?
- How can we automatically generate test cases from models?
- How can we provide *incremental* support for verification and validation?
- How can we deal with partially incomplete and inconsistent models?
- How can we support formal verification of models?
- How can we address validation and verification in a multi-model world?

Process support. Model-driven engineering encompasses many more activities than merely modelling. One important aspect that is often overlooked by the scientific community is *process support*. This gives rise to a number of essential questions:

- Which processes should be used for MDE?
- How should existing processes embrace MDE?
- How should we teach and educate people in adopting MDE technology?
- How can we incorporate the MDE environment in the MDE process?

Fuzzy modelling. Models are not always complete and sometimes inconsistencies need to be tolerated. This gives rise to specific questions like:

- How can we deal with modelling in presence of uncertainty?
- How can we deal with models that are imperfect or ambiguous?
- How can we reason about models that are incomplete or inconsistent?
- How can we cope with imprecision of models?

Industrial adoption. This topic will be discussed in section 4.1.

Formal foundations. This topic will be discussed in section 4.2.

Scaleability issues. This topic will be discussed in section 4.3.
Model consistency and co-evolution. This topic will be discussed in section 4.4.

As a general kind of meta-challenge, it was suggested by one of the participants that we need to be aware more of relevant past research (possibly in other software engineering domains), rather than trying to reinvent the wheel.

4 Discussion of Breakout Groups

Because it was not possible to explore all of the above challenges in detail during the workshop, we decided to break out into 4 different groups, each one focusing on a particular set of challenges that were considered to be important by the majority of participants.

4.1 Industrial Adoption

The "engineering" aspect of model-driven engineering implies that research in MDE is useless without having industrial adoption. The MDE community could benefit a lot from concrete industrial use cases, both positive and negative ones.

From the positive side, it would be good to learn which companies have successfully adopted MDE technology, and what was the reason of this success: Which tools, techniques and processes have been used, and what was the added value brought by MDE? Several participants mentioned examples of such success stories. For example, in the automotive industry, the use of MDE technology is standard practice. The same appears to be true for real-time and embedded systems. Also in the area of web application development there are various approaches that support MDE (e.g., AndroMDA). Finally, there were examples of the use of MDE in the telecommunications and insurance domains.

The opposite question was also discussed. Can we find failures of the use of MDE in industry, and the reasons underlying these failures? Similarly, can we find reasons why some software companies are not using MDE? Some interesting points were raised when discussing about these questions. First of all, there is a significant technological threshold and learning curve before you can actually use MDE in industry. Therefore, using MDE technology may not be cost effective for many industrial application scenarios. The argument was also made that, although some companies do not use MDE, they do use models for communication. Finally, while some companies may be interested in using MDE technology, it may not be possible if they still have a large legacy code base available that has been around for decades, and has been developed in "old" technology (e.g, COBOL code) that may be too costly or too hard to migrate.

One thing that appeared to be common to all discussed industrial use cases was the use of domain-specific modelling languages. This seems to indicate that MDE works well for specific problems in specific domains, and that the use of a *universal* modelling language (e.g., UML) may possibly not work well in an industrial setting.

We also discussed how MDE can bring value to industry. In some of the industrial cases discussed, MDE was used because of its ability to formally specify and analyse (part of) the system. This leads to a reduction in ambiguity and improved quality. Other potential advantages are cost reduction, productivity improvement, easier maintenance, and detection of problems and inconsistencies in the software system early in the life cycle. Of course, it is essential to keep in mind that any of these potential advantages should not be detrimental to the performance of the software system to be produced.

The main issue with the above is that we need to convince industry that there is actually added value of MDE, and we should come up with a deployment model to enable the adoption of MDE technology in industry. The only way this can be done is by performing convincing empirical case studies of the use of MDE in industry. Obviously, to be able to perform such studies, we need to be able to obtain detailed data about MDE practice from industry itself. In practice, it turns out to be very difficult to obtain industrial data, and to transfer technology and research results from academia to industry. Participants of the breakout group basically agreed that the only viable way to achieve this is by direct contact between the research community and industry. One way to establish such contact is via exchange programmes in which PhD students spend a couple of months in a company to understand the process used and the particular activities that are amenable to automation via MDE technology, as well as to raise awareness of industry in the benefits of MDE. Other possibilities are the use of dedicated industrial education and training programmes.

4.2 Formal Foundation

Verification and validation is an important research theme in the MDE community. To be able to verify and validate models and model transformations, a formal foundation is a necessity. The first challenge identified by the participants of this breakout group was to integrate formal verification tools into modelling environments. This needs to be done in such a way that the user of the modelling environment does not need to have expertise in the different formalisms and techniques used for verification. The feedback of these verification tools needs to be formulated in a language or formalism that the end user of the environment is familiar with.

To realise this smooth integration, the participants of the breakout group agreed that transformations from modelling languages to formal verification and analysis models need to be defined. The definition of transformations triggers several interesting challenges. How to define such transformations? How to prove correctness of model transformations, especially if the source models are informal? And how to proof that the transformation is correct? It is possible that the transformation of an informal model to a formal model is correct by construction, since the main goal of such semantic mapping is to define a precise semantic meaning for the concepts of the informal model. All participants of the breakout group agreed that first of all the notion of correctness needs to be defined, because many variations of correctness definitions exist in state-of-the-art literature. Once models are transformed into a certain formalism and verification of properties has been executed in this formalism, feedback needs to be given to the end user and incorporated into the source model. The question arises on how to reinterpret these results in the source models and tools.

There is a lot of existing work on using formalisms to support model-driven engineering. Examples are graph transformation theory, algebraic specifications, model checking, logic-based approaches and SAT solvers, and category theory. In the programming language area, operational, denotational and axiomatic semantics exist. These different approaches are useful for investigating different kinds of properties. The participants also recognised that different semantics and formalisms may be necessary at different phases in the development life cycle, and at at different levels of abstraction, since a single formalism may not fit all the models describing various aspects of a complex system. This gives rise to an interesting challenge: how to relate these levels and how to define the relationships between them. The participants posed the question whether it is necessary to define relations between the different formalisms. As a complex system is gradually being modelled using a multitude of often large models, and regularly extended and adapted, incremental verification and validation and scalability of the verification and validation tools become key challenges for MDE.

Precisely defining domain-specific modelling languages was another discussion topic. This raises the question how to help developers design good modelling languages that guarantee useful properties to the users of these languages. Reusability was recognised as a key issue in modelling language design. In analogy with design patterns, the participants propose to identify and define patterns and anti-patterns for designing modelling languages. The main challenge that was identified is to define domain-specific modelling languages that enable and enforce model correctness by construction.

All participants agreed that, despite the multitude of existing formalisms and experiments to use them in MDE, a lot of research still needs to be done. This is especially true for tool support for verification and validation, as well as support for defining well-designed modelling languages. A goal-driven approach for MDE was suggested, by focusing on the question of what needs to be the added value of the languages, techniques, and tools?

4.3 Scaleability

Scaleability is a general problem in software engineering. Many software engineering research areas are struggling to cope with scaleability issues, and a large research effort has already been spent to develop solutions for overcoming scaleability problems. The MDE community must therefore focus on (1) the kind of scalability issues that are intrinsic for MDE; (2) elements about MDE do not scale well and the underlying reasons thereof; and (3) specific scalability problems for MDE that cannot be addressed by existing solutions from other software engineering domains.

Concerning the intrinsic type of scalability needed for MDE, one of the main problems is that MDE has to be able to cope with very large models in order to model systems of systems and Ultra-Large-Scale (ULS) systems. These models have to be constructed, transformed, merged, and used as a base for code generation. So one could try to develop solutions for optimising these activities in order to use them adequately on large models. However, often solutions for one type of activity can be rather different than those necessary for other types of activities. The question arises whether generic optimisation solutions can be developed for MDE activities. In addition, one must be aware that the time to load huge models is often greater than the time needed for checking, merging or transforming such models.

Elements that can cause scalability problems in an MDE approach are, among others, multi-site collaborative development, complexity of algorithms manipulating models, computer resources needed to manage huge models, and technical limitations of the used notations and tools (concerning support for modularity, concurrent access, distribution, etc.). In addition, the accidental complexity of underlying MDE technology should be reduced.

Mechanisms and techniques from other software engineering domains could be useful for solving MDE scaleability issues. From the programming community, techniques such as modular engineering principles, incremental processing, caches, and indices could be beneficial for MDE. Further solutions can come from logic inference engines for model checking, and high performance computing for optimisation techniques. The participants assessed that there are known solutions to all the problems they thought of, however, the issue is to generalise them for MDE. In addition, the design of modelling languages seems not always to respect known scaling problems in concrete languages.

4.4 Model Evolution and Inconsistency Management

Models do not appear after a big bang, but are often developed by different persons in a distributed setting using different modelling languages. Such multi-user distributed setting, combined with the usage of different modelling languages to model a system, can cause inconsistencies in and between models. Models evolve and so do their meta-models. The major challenge is to assess the impact of change of a model or meta-model on the other models and meta-models. The challenge increases if models are distributed. The participants of this breakout group propose – as a small step towards a solution – to categorise the different change types and the possible ways to resolve the inconsistencies introduced by the changes.

Models are built using a variety of domain-specific modelling languages. The question arises how to develop DSMLs can be *efficiently* extended, adapted or customised. The term *efficiency* is strongly related to traditional quality measures. More effort should go to a DSML process. One possible solution would be to have rapid prototyping for building DSMLs. Prototyping has the advantage of giving continuous, incremental feedback. In the context of evolution the question arises how a DSML evolves from version $n-1$ to version n and what happens with the existing models adhering to the DSML version $n-1$?

Other challenges that were identified are: how to deal with long-lived models and legacy models? How to maintain models? How to avoid model erosion? How to support long-lived software intensive systems that have been generated using MDE? The participants stated that more effort should go to a model-driven development process.

Once inconsistencies are identified in or between models, the question arises how to deal with these inconsistencies. Model-driven development environments need built-in support for inconsistency handling and resolution. Built-in support for inconsistency detection and handling is also needed in versioning tools. In a model-driven software development process, handling inconsistencies at model level will also affect the (generated) code. As such, an important research question is how to support model-code synchronisation and round-trip engineering.

Formalisms and techniques to detect, handle and resolve inconsistencies can be based on formal verification techniques used in other software engineering domains such as programming language engineering, but also on formalisms and techniques used in database engineering and artificial intelligence.

The participants also discussed the question how collaboration in this field can be improved. They suggest two possible approaches. First, the development of an ontology for the consistency/evolution area, and second, a survey that unifies the different existing perspectives.

5 Past and Future of MDD

As one of the outcomes of the workshop, besides the current workshop report, each breakout group prepared a single question that was passed to the panelists of the MoDELS'08 panel on the *"Past and Future of MDD"*. These questions were:

– How can we better understand the software process and activities that companies use and improve them with MDE technology?
– Can incremental model verification and model checking contribute to successful adoption of MDE?
– Are there scalability problems that are specific to MDE, and how can we address them?
– How can we deal with co-evolution of models, meta-models and transformations in a distributed multi-developer environment?

6 Conclusions

During the workshop, a large number of challenges for MDE have been identified, covering a broad range of topics that are important for the successful application of MDE in practice. The organisers hope that the workshop results help to identify an MDE research agenda, to define the roadmap for future MDE research, and to inspire researchers for tackling important problems and develop novel and adequate solutions.

Acknowledgements

This workshop was organised in the context of three research projects:

- the research project "Modelling, Verification and Evolution of Software (MoVES)", an IAP-Phase VI *Interuniversity Attraction Poles Programme* funded by the Belgian State, Belgian Science Policy, http://moves.vub.ac.be
- the research project "Model-Driven Software Evolution", an *Action de Recherche Concertée* financed by the Ministère de la Communauté française - Direction générale de l'Enseignement non obligatoire et de la Recherche scientifique.
- the EUREKA-ITEA2 research project "Evolutionary Validation, Verification and Certification (EVOLVE)", partially funded by the Flemish government institution IWT (Institute for the Promotion of Innovation by Science and Technology in Flanders), http://www.evolve-itea.org

We thank all workshop participants for the lively discussions and the useful feedback we received. We thank the workshop programme committee members for their helpful reviews: Jean Bézivin, Xavier Blanc, Dirk Deridder, Gregor Engels, Vincent Englebert, Robert France, Dragan Gasevic, Sébastien Gérard, Wouter Joosen, Anneke Kleppe, Jochen Küster, Richard Paige, Ivan Porres, Laurent Rioux, Bernhard Rumpe, and Hans Vangheluwe.

References

1. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: FOSE 2007: 2007 Future of Software Engineering, pp. 37–54. IEEE Computer Society Press, Washington (2007)
2. Van Baelen, S., Van Der Straeten, R., Mens, T. (eds.): ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering, VUB, UMH, K.U.Leuven (2008)
3. Mohagheghi, P., Fernandez, M., Martell, J., Fritzsche, M., Giliani, W.: MDE adoption in industry: Challenges and success criteria. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering, pp. 5–9 (2008)
4. Kolovos, D.S., Paige, R.F., Polack, F.A.: Scalability: The holy grail of model driven engineering. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering, pp. 10–14 (2008)
5. Posse, E., Dingel, J.: A foundation for MDE. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering, pp. 15–19 (2008)
6. Rivera, J.E., Romero, J.R., Vallecillo, A.: Behavior, time and viewpoint consistency: Three challenges for MDE. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering, pp. 20–24 (2008)

7. Wagelaar, D.: Challenges in bootstrapping a model-driven way of software development. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering, pp. 25–30 (2008)
8. Cabot, J., Clarisó, R.: Uml/ocl verification in practice. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering, pp. 31–35 (2008)
9. Cabot, J., Yu, E.: Improving requirements specifications in model-driven development processes. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering, pp. 36–40 (2008)
10. Rech, J., Bunse, C. (eds.): Model-Driven Software Development: Integrating Quality Assurance. Information Science Reference (2008)

The Grand Challenge of Scalability for Model Driven Engineering

Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack

Department of Computer Science, University of York, UK
{dkolovos,paige,fiona}@cs.york.ac.uk

Abstract. Scalability is particularly important for the adoption of Model Driven Engineering (MDE) in an industrial context. The current focus of research in MDE is on declarative languages for model management, and scalable mechanisms for persisting models (e.g., using databases). In this paper we claim that, instead, modularity and encapsulation in modelling languages should be the main focus. We justify this claim by demonstrating how these two principles apply to a related domain – code development – where the issue of scalability has been addressed to a much greater extent than in MDE.

1 Introduction

The adoption of MDE technologies in an industrial context involves significant benefits but also substantial risks. Benefits in terms of increased productivity, quality and reuse are easily foreseeable. On the other hand, the most important concerns raised of MDE are those of scalability [1], the cost of introducing MDE technologies to the development process (training, learning curve) and longevity of MDE tools and languages. To our perception, the latter two concerns (cost of induction and longevity) are not preventive for the adoption of MDE; however scalability is what is holding back a number of potential adopters.

2 Scalability in MDE

Large companies typically develop complex systems, which require proportionally large and complex models that form the basis of representation and reasoning. Moreover, development is typically carried out in a distributed context and involves many developers with different roles and responsibilities. In this context, typical exploratory questions from industrial parties interested in adopting MDE include the following:

1. *In our company we have huge models, of the order of tens of thousands of model elements. Can your tool/language support such models?*
2. *I would like to use model transformation. However, when I make a small change in my (huge) source model, it is important that the change is incrementally propagated to the target model; I don't want the entire target model to be regenerated every time.*

3. (similarly) *I would like to use code generation. However, when I make a small change in my (huge) model I don't want all the code to be regenerated.*
4. *In my company we have many developers and each manages only a specific part of the model. I would like each developer to be able to check out only a part of the model, edit it locally and then merge the changes into the master copy. The system should also let the developers know if their changes are in conflict with the rest of the model or with changes done by other developers.*

Instead of attempting to answer such questions directly, we find it useful to consider analogies with a proven and widely used environment that addresses those problems in a different – but highly relevant – domain. The domain is code development and the environment is the well known and widely used Eclipse Java Development Tools (JDT).

As a brief overview, JDT provide an environment in which developers can manage huge code-bases consisting of (tens of) thousands of Java source code files (concern 1). JDT supports incremental consistency checking and compilation (concerns 2,3) in the sense that when a developer changes the source code of a particular Java class, only that class and any other classes affected by the change – as opposed to all the classes in the project or the workspace – are re-validated and re-compiled. Finally, JDT is orthogonal to version control, collaborative development (concern 4), and multi-tasking tools such as CVS and SVN and Mylyn.

3 Managing Volume Increase

As models grow, tools that manage them, such as editors and transformation engines, must scale proportionally. A common concern often raised is that modelling frameworks such as EMF [2] and widely-used model management languages do not scale beyond a few tens of thousands of model elements per model. While this is a valid concern, it is also worth mentioning that the Java compiler does not allow Java methods the body of which exceed 64 KB, but in the code-development domain this is rarely a problem.

The reason for this asymmetry in perception is that in code development, including all the code of an application in a single method/file is considered – at least – bad practice. By contrast, in modelling it is deemed perfectly *reasonable* to store a model that contains thousands of elements in a single file. Also, it is *reasonable* that any part of the model can be hard-linked with an ID-based reference to any other part of the model.

To deal with the growing size of models and their applications, modelling frameworks such as EMF support lazy loading and there are even approaches, such as Teneo [3] and CDO [4], for persisting models in databases. Although useful in practice, such approaches appear to be temporary workarounds that attempt to compensate for the lack of encapsulation and modularity constructs in modelling languages. In our view, the issue to be addressed in the long run is not how to manage large monolithic models but how to separate them into smaller modular and reusable models according to the well understood principles defined almost 40 years ago in [5], and similarly to the practices followed in code development.

4 Incrementality

In the MDE research community, incrementality in model management is sought mainly by means of purely declarative model transformation approaches [6,7]. The hypothesis is that a purely declarative transformation can be analysed automatically to determine the effects of a change in the source model to the target model. Experience has demonstrated that incremental transformations are indeed possible but their application is limited to scenarios where the source and target languages are similar to each other, and the transformation does not involve complex calculations.

JDT achieves incrementality without using a declarative language for compiling Java source to bytecode; instead it uses Java which is an imperative language. The reason JDT can achieve incremental transformation lies mainly the design of Java itself. Unlike the majority of modelling languages, Java has a set of well-defined modularity and encapsulation rules that, in most cases, prevent changes from introducing extensive ripple effects.

But how does JDT know what is the scope of each change? The answer is simple: it is hard-coded (as opposed to being automatically derived by analysing the transformation). However, due to the modular design of the language, those cases are relatively few and the benefits delivered justify the choice to hard-code them. Also it is worth noting that the scope of the effect caused by a change is not related only to the change and the language but also to the intention of the transformation. For example, if instead of compiling the Java source code to bytecode we needed to generate a single HTML page that contained the current names of all the classes we would unavoidably need to re-visit all the classes (or use cached values obtained earlier).

5 Collaborative Development

As discussed in Section 2, a requirement for an MDE environment of industrial strength is to enable collaborative development of models. More specifically, it is expected that each developer should be able to check out an arbitrary part of the model, modify it and then commit the changes back to the master copy/repository. Again, the formulation of this requirement is driven by the current status which typically involves constructing and working with large monolithic models. With enhanced modularity and encapsulation, big models can be separated into smaller models which can then be managed using robust existing collaborative development tools such as CVS and SVN, augmented with model-specific version comparison and merging utilities such as EMF Compare [8]. Given the criticality of version control systems in the business context, industrial users are particularly reluctant to switching to a new version control system[1]. Therefore, our view is that radically different solutions, such as dedicated model repositories,

[1] Evidence of this is that CVS which was introduced in the 1980s is still the most popular version control system despite its obvious limitations compared to newer systems such as SVN

that do not build on an existing robust and proven basis are highly unlikely to be used in practice.

6 Modularity in Modelling Languages

The above clearly demonstrate the importance of modularity and encapsulation for achieving scalability in MDE. There are two aspects related to modularity in modelling: the design of the modelling language(s) used and the capabilities offered by the underlying modelling framework. In this section we briefly discuss how each of those aspects affect modularity and envision desirable capabilities of modelling frameworks towards this direction.

6.1 Language Design

With the advent of technologies such as EMF and GMF [9], implementing a new domain-specific modelling language and supporting graphical and textual editors is a straightforward process and many individuals and organizations have started defining custom modelling languages to harvest the advantages of the context-specific focus of DSLs. When designing a new modelling language, modularity must be a principal concern. The designers of the language must ensure that large models captured using the DSL can be separated into smaller models by providing appropriate model element *packaging* constructs. Such constructs may not be part of the domain and therefore they are not easily foreseeable. For example, when designing a DSL for modelling relational databases, it is quite common to neglect *packaging*, because relational databases are typically a flat list of tables. However, when using the language to design a database with hundreds of tables, being able to group them in conceptually coherent packages is highly important to the manageability and understandability of the model.

6.2 Modelling Framework Capabilities

In contemporary modelling frameworks there are three ways to capture relationships between two elements in a model: containment, hard references and soft references. Containment is the natural relationship of one element being a composite part of another, a hard reference is a unique-ID-based reference that can be resolved automatically by the modelling framework and a soft reference is a reference that needs an explicit resolution algorithm to navigate [10].

To enable users to split models over multiple physical files, contemporary modelling frameworks support cross-model containment (i.e. the ability of a model element to contain another despite being stored in different physical files). With regard to hard and soft non-containment references, hard references are typically proffered because they can be automatically resolved by the modelling framework and thus, they enable smooth navigation over the elements of the model with languages such as OCL and Java. Nevertheless, in our view hard references are particularly harmful for modularity as they increase coupling between different parts of the model and prevent users from working independently on different parts. On the other hand, soft references enable clean separation of model fragments but require custom resolution algorithms which have to be implemented from scratch each time.

To address this problem, we envision extensions of contemporary modelling frameworks that will be able to integrate resolution algorithms so that soft references can be used, and the efficient and concise navigation achievable with languages such as OCL can still be performed.

7 Conclusions

In this paper we have demonstrated the importance of modularity and encapsulation for achieving scalability in MDE. We have identified two main problems: neglect of modularity constructs during the design of modelling languages and extensive use of ID-based references that lead to high coupling between different parts of the model. With regard to the first issue we have been working on preparing a set of guidelines for the design of scalable and modular DSLs and expect to report on this soon. The second issue is quite more complex and we plan to elaborate and prototype a solution based on EMF and Epsilon [11] in the near future.

Acknowledgements

The work in this paper was supported by the European Commission via the MODELPLEX project, co-funded by the European Commission under the "Information Society Technologies" Sixth Framework Programme (2006-2009).

References

1. Warmer, J., Kleppe, A.: Building a Flexible Software Factory Using Partial Domain Specific Models. In: Proc. 6th OOPSLA Workshop on Domain-Specific Modeling, Portland, Oregon, USA (October 2006)
2. Eclipse Foundation. Eclipse Modelling Framework, `http://www.eclipse.org/emf`
3. Eclipse Foundation. Teneo (2008), `http://www.eclipse.org/modeling/emft/?project=teneo`
4. Eclipse Foundation. CDO (2008), `http://www.eclipse.org/modeling/emft/?project=cdo`
5. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of ACM 15(12), 1053–1058 (1972)
6. Hearnden, D., Lawley, M., Raymond, K.: Incremental model transformation for the evolution of model-driven systems. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 321–335. Springer, Heidelberg (2006)

7. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling, 1619–1374 (March 2008)
8. Eclipse Foundation. EMF Compare (2008), `http://www.eclipse.org/modeling/emft/?project=compare`
9. Eclipse GMF - Graphical Modeling Framework, Official Web-Site, `http://www.eclipse.org/gmf`
10. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: Detecting and Repairing Inconsistencies Across Heterogeneous Models. In: Proc. 1st IEEE International Conference on Software Testing, Verification and Validation, Lillehammer, Norway, pp. 356–364 (April 2008)
11. Extensible Platform for Specification of Integrated Languages for mOdel maNagement (Epsilon), `http://www.eclipse.org/gmt/epsilon`

MDE Adoption in Industry: Challenges and Success Criteria

Parastoo Mohagheghi[1], Miguel A. Fernandez[2], Juan A. Martell[2], Mathias Fritzsche[3], and Wasif Gilani[3]

[1] SINTEF, P.O. Box 124-Blindern, N-0314 Oslo, Norway
parastoo.mohagheghi@sintef.no
[2] Telefónica Research & Development, Valladolid, Spain
{mafg@tid.es,jamartell}@gfi-info.com
[3] SAP Research CEC Belfast, United Kingdom
{mathias.fritzsche,wasif.gilani}@sap.com

Abstract. Model-Driven Engineering has been promoted for some time as the solution for the main problem software industry is facing, i.e. complexity of software development, by raising the abstraction level and introducing more automation in the process. The promises are many; among them improved software quality by increased traceability between artifacts, early defect detection, reducing manual and error-prone work and including knowledge in generators. However, in our opinion MDE is still in the early adoption phase and to be successfully adopted by industry, it must prove its superiority over other development paradigms and be supported by a rich ecosystem of stable, compatible and standardized tools. It should also not introduce more complexity than it removes. The subject of this paper is the challenges in MDE adoption from our experience of using MDE in real and research projects, where MDE has potential for success and what the key success criteria are.

Keywords: Model-driven engineering, challenges, domain-specific modeling, performance engineering, traceability.

1 Introduction

Today's software systems are complex in nature; the size has been growing because of the increased functionality, heterogeneity is also becoming a bigger concern as systems are built from several systems or include legacy code, systems are distributed over multiple sites and there are new requirements such as dynamicity and autonomy (self-* properties, for example self-healing). Handling each of these challenges requires specific approaches which often include domain-specific knowledge and solutions. However, based on the experience gained from multiple domains and projects, some solutions may be identified as beneficial to complex software development in general.

Model-Driven Engineering (MDE) is an approach built upon many of the successful techniques applied in software engineering: It can be characterized by: a) raising the abstraction level by hiding platform-specific details ; b) taking advantage of models in all the phases of software development to improve understanding; c) developing domain-specific languages and frameworks to achieve domain appropriateness; and d) taking advantage of transformations to automate repetitive work and improve software quality [6]. These are all techniques useful for complex system development and therefore one may expect rapid adoption of the paradigm by industry. So far, we cannot see such wide adoption, as also confirmed by a review of industrial experiences presented in [7]. In fact, and based on the model of technology adoption life cycle presented in [8], we think that MDE is still in the early adoption stage. Early adopters do not rely on well-established references in making their buying decisions, preferring instead to rely on their own intuition and vision. However, they are keys to opening up any high-tech market segment. To be accepted by the majority, the industry must gain confidence on the promises of MDE and have access to proper tools and experts.

There are many challenges in complex system development, such as managing requirements, which MDE is not a direct answer to, but it might facilitate their handling by providing mechanisms for easy traceability between artifacts. There are also challenges such as dealing with legacy code that may be difficult to handle and must be either worked around or, better yet, integrated into the MDE approaches. But there are challenges that MDE may provide an answer to based on the MDE core practices (such as extensive modeling and the usage of transformations) as discussed in [6].

The European research projects MODELWARE[1] and its continuation MODELPLEX[2] have focused on MDE approaches and tools with the goal of making them suitable for complex system development. Some of the companies involved in these projects have experience from applying MDE in real projects while others think that MDE is not yet mature enough to be taken from research projects to industry production. This paper therefore elaborates on where we can expect added value from MDE and what the barriers are from experiences gained in the context of these projects. In the remainder of this paper we discuss industry expectations and experience in Sections 2 and 3 and conclude our discussion in Section 4.

2 SAP Experience

SAP has already started working towards applying MDE concepts, and currently employs models in various stages of business application development. The tool called NetWeaver BPM within the Composition Environment [10] is one example where MDE concepts are applied for efficient development of Composite Applications. Composite Applications are self-contained applications that combine loosely coupled services (including third party services) with their own business logic, and thereby provide user centric front-end processes that transcend functional boundaries, and are completely independent from the underlying architecture, implementation and software lifecycle. With Composition Environment even the non-technical users, such as business domain experts, consultants, etc., having no programming skills, are able to model and deploy customized applications suited to their specific business requirements.

[1] http://www.modelware-ist.org/
[2] http://www.modelplex-ist.org/

Based on our experience [5] with the currently employed MDE tools for business processes, such as the Composition Environment, we identified the general need of supporting non-technical users with regards to non-functional requirements, such as the impact of their design decisions on performance, etc. Within the context of performance engineering, for instance, such a support means guidance towards better design / configuration that actually meets the timelines, and optimized resource mapping against each activity in the business process.

We implemented such performance related decision support as an extension of MDE. By implementing this extension, named Model-Driven Performance Engineering (MDPE), we realized the need for supporting requirements with respect to non-functional aspects, especially performance. The implementation of MDPE heavily uses the MDE concepts such as meta-modeling, transformations, model weaving and mega-modeling. For instance, ten different meta-modeling languages are employed in order to make the process usable for a number of domain-specific modeling languages. During the implementation of MDPE, we recognized that the application of MDE concepts enabled us to focus on the creative tasks of development rather than repetitive coding. For instance, code generation for our meta-models saved us significant development effort. The only place where a significant amount of coding effort was required was for the integration of MDPE into the existing tool infrastructure.

Meta-model extension is the generally employed technique for model annotations, such as done with profiles in the case of UML [3]. However, this is not applicable while dealing with the proprietary models. The application of model weaving enabled us a high degree of flexibility as we are able to annotate any kind of proprietary model with the help of a generic editor [3]. Higher-order transformations are used to enable traceability in our approach [4]. Additionally, mega-modeling enables us to locate our model artifacts, such as the tracing models related to the models in our transformation chain [1].

As for the challenges, we experienced that MDE concepts are on the one hand very systematic and efficient, but on the other hand also difficult to understand for developers as they require quite a high level of abstraction and training. Also, the MDE tool support is sometimes not mature enough. Especially the available tooling to define model transformation chains lacks capabilities of modern IDEs (Integrated Development Environments), which could decrease the development time for model transformations significantly.

Concluding, based on the experiences gained with the development of MDPE, we are optimistic regarding the capabilities of MDE in case the tool support improves, and the MDE community meets the challenges associated with the MDE process, such as providing support for dealing with non-functional aspects of system development.

3 Telefónica Experience

In [2], we have discussed the experience of Telefónica in moving from a code-centric to a model-centric software development. Earlier efforts in modeling failed due to the complexity of UML, the lack of proper tools and the inability to maintain models in synch with code, among other issues. Due to the above problems with UML, we decided to develop our own programming tools and frameworks addressing the problem domain. But without any industry standards to rely on, this approach had no future in the long term and was also difficult to use for non-technical staff, such as telecom domain experts, as it did not have the required abstraction level.

This was an experience from eight years ago, but not so many things seem to have fundamentally changed. What we look for is a domain-specific modeling (DSM) language integrated in a development environment that will permit the modeling of our basic domain concepts, such as interfaces, devices, networks, protocols and services. We also emphasize adhering to current industry standards in the domain, since we now look for a domain-specific solution, not a company-wide solution. Other requirements are: a) the ability to model in multiple abstraction levels, hiding details as desired; b) the integration of model verification tools based on OCL or other constraint languages and c) the composition / weaving of the models at run time to reflect the changes in the network's operational status. Some of these approaches are further discussed in [9].

In the road toward these objectives we foresee numerous challenges. First of all, the UML standard has evolved but, with this evolution, the syntax has become even more complex and the necessary supporting mechanisms and tools for dealing with this added complexity are not yet available. Even something as conceptually simple as exporting a UML diagram from one tool to another has not been accomplished yet with ease. On the other hand, developing a DSM solution requires high skills related to meta-modeling and tool development. Also a big concern with Domain-Specific Languages (DSLs) is getting the people in that domain to agree upon a standard syntax. Another challenge is having that DSL interact properly with anything outside of its domain, having a different underlying syntax to that of other languages.

Model synchronization (for example applying multiple profiles to a source model) and roundtrip engineering are yet to be addressed successfully and mechanisms for dealing with very large and complex models, such as hierarchical models, traceability and model management in general are also in an inception phase right now, at least regarding to the aspect of tool support. The evolution of meta-models, in a business as dynamic as ours, is also a big concern and tools have much to improve in order to adequately manage variability at meta-model level and not only at model level. All these features are important to make a full-fledged MDE process work in complex, real-life projects.

Another challenge for organizations wanting to get started in MDE, closely related with the previous idea of managing all these artifacts, is that they may end up dealing with more complexity than anticipated at first. From our experience in the field we have gotten the impression that, if not adequately managed, the development of complex systems with MDE gets treated with more complexity. The underlying problem here is: are the techniques for handling complexity in danger of making the software engineering process itself too complex? To adequately address complexity we have to substitute it for something simpler not for something different but equally complex.

It is our opinion also that there are some basic milestones a new technology has to go through for it to be considered mainstream. To start with, we need a proper context for it to flourish and be nurtured in. The fabric of this context is made of the proper professionals with the proper knowledge and expertise and supporting material which helps in turn to create these professionals. We are seeing shortcomings in this regard so far. The community is in fact there and growing but perhaps it is not reaching critical mass yet. We also see a gap between the academic and industrial worlds that needs to be bridged. In the past, new paradigms have been promoted by well-known professionals lending credibility and raising interest in the new approach. This has to be accompanied by the development of high-quality literature, tutorials and proper material to draw new professionals in.

The main question that an organization has to ask itself is “do I really need MDE?” The second question relates with its ability to adapt its processes to the ones needed from an MDE point of view (partially discussed also in [2]), adapt their staff to new ways of looking at problems and create new layers of software development supporting all the aspects MDE has to offer. Companies may be reluctant to change either their structure or part of it.

To conclude, it is worth mentioning that, apart from software factories for product line engineering (PLE), we have not seen clear evidence of other good candidates for MDE to be fully applied to, as a complete lifecycle solution. We feel that it can be partially applied, though, to some other scenarios like large-scale integration of heterogeneous systems, as it is the case with Telefónica’s Operating Support Systems (OSS), area in which we hope to start making some progress in the short term with Model-Based Testing (MBT).

4 Conclusions

Probably most companies cannot take the risk of adopting MDE end-to-end in large-scale projects from scratch. They should look for areas of improvement and take the approach incrementally and integrated with their own development environment. This is also the best way to train people. There is an initial high cost related to developing or adopting tools and transformations. MDE is a long-term investment and needs customization of environment, tools and processes, and training. For companies that have a product line, MDE can pay off since this cost is amortized over several projects. For one-of–a-kind projects this will not pay in most cases. Despite differences in domain and the type of systems developed in the two companies, there are common challenges as described in this paper. The most important one is the complexity of developing an MDE environment tailored to the company needs. This environment requires:

- Developing proper languages for communication between technical and non-technical experts and for modeling various aspects. One of the successes of MDE lies in bridging the gap between technical and non-technical experts. The major challenge here is to have the required language engineering expertise since creating own profiles or meta-models are difficult and for complex systems we probably need several languages. Hence more domain-specific meta-models and profiles are needed that are supported by tools and may be reused. The current tools for developing meta-models and editors are not user friendly, the learning curve is steep and the documentation and support is not satisfactory.
- Several tools are required for modeling, model-to-model and model-to-text transformation, verification and simulation, and other tools to store, reuse and compose models. There is no tool chain at the moment and companies must integrate several tools and perform adaptation themselves.

Both of the above requirements put a high burden on companies that traditionally used third-party tools for modeling and performed programming by hand. Training is another major challenge here. We see advantages in gradual introduction and support by management, as well as in the creation of teams of experts that can give support and create the necessary tools for MDE adoption in the whole company.

Acknowledgments. Part of the ideas presented in this paper are based on conclusions obtained in the MODELPLEX project (IST-FP6-2006 Contract No. 34081), co-funded by the European Commission as part of the 6th Framework Program.

References

1. Barbero, F., Jouault, J.: Model Driven Management of Complex Systems: Implementing the Macroscope's Vision. In: 15th ECBS 2008, pp. 277–286. IEEE Press, Los Alamitos (2008)
2. Fernandez, M.: From Code to Models: Past, Present and Future of MDE Adoption in Telefónica. In: 3rd Europen Workshop From Code Centric to Model Centric Software Engineering: Practices, Implications and Return on Investment (C2M), co-located with ECMDA 2008, pp. 41—51 (2008)
3. Fritzsche, M., Johannes, J., et al.: Systematic Usage of Embedded Modelling Languages in Model Transformation Chains. In: The Software Language Engineering Conference, SLE 2008 (accepted, 2008)
4. Fritzsche, M., Johannes, J., Zschaler, S., Zherebtsov, A., Terekhov, A.: Application of Tracing Techniques in Model-Driven Performance Engineering. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095. Springer, Heidelberg (2008)
5. Fritzsche, M., Gilani, W., Fritzsche, C., Spence, I.T.A., Kilpatrick, P., Brown, T.J.: Towards utilizing Model-Driven Engineering of Composite Applications for Business Performance Analysis. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 369–380. Springer, Heidelberg (2008)
6. Mohagheghi, P.: Evaluating Software Development Methodologies based on their Practices and Promises. In: Proc. Somet 2008: New Trends in Software Methodologies, Tools and Techniques, pp. 14–35. IOS Press, Amsterdam (2008)
7. Mohagheghi, P., Dehlen, V.: Where is the Proof? A Review of Experiences from Applying MDE in Industry. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 432–443. Springer, Heidelberg (2008)
8. Moore, G.A.: Crossing the chasm: Marketing and Selling High-Tech Products to Mainstream Customers, 2nd edn. HarperBusiness Essentials (2002)
9. Pickering, B., Fernandez, M., Castillo, A., Mengusoglu, E.: A Domain-Specific Approach for Autonomic Network Management. In: van der Meer, S., Burgess, M., Denazis, S. (eds.) MACE 2008. LNCS, vol. 5276. Springer, Heidelberg (2008)
10. Snabe, J.H., Rosenber, A., Møller, C., Scavillo, M.: Business Process Management: The SAP Roadmap. SAP Press (2008) ISBN 978-1-59229-231-8

Behavior, Time and Viewpoint Consistency: Three Challenges for MDE

José Eduardo Rivera[1], José Raul Romero[2], and Antonio Vallecillo[1]

[1]Universidad de Málaga Spain
[2]Universidad de Córdoba Spain
{rivera,av}@lcc.uma.es, jrromero@uco.es

Abstract. Although Model Driven Software Development (MDSD) is achieving significant progress, it is still far from becoming a real Engineering discipline. In fact, many of the difficult problems of the engineering of complex software systems are still unresolved, or simplistically addressed by many of the current MDSD approaches. In this position paper we outline three of the outstanding problems that we think MDSD should tackle in order to be useful in industrial environments.

1 Introduction

Although both MDSD and MDA have experienced significant advances during the past 8 years, some of the key difficult issues still remain unresolved. In fact, the number of engineering practices and tools that have been developed for the industrial design, implementation and maintenance of large-scale, enterprise-wide software systems is still low — i.e. there are very few real *Model-Driven Engineering* (MDE) practices and tools. Firstly, many of the MDSD processes, notations and tools fall apart when dealing with large-scale systems composed of hundred of thousands of highly interconnected elements; secondly, MDE should go beyond conceptual modeling and generative programming: it should count on mature tool-support for automating the design, development and analysis of systems, as well as on measurable engineering processes and methodologies to drive the effective use of all these artifacts towards the predictable construction of software systems. In particular, engineering activities such as simulation, analysis, validation, quality evaluation, etc., should be fully supported.

We are currently in a situation where the industry is interested in MDE, but we can easily fail again if we do not deliver (promptly) anything really useful to them. There are still many challenges ahead, which we should soon address in order not to lose the current momentum of MDE.

In this position paper we focus on three of these challenges. Firstly, the specification of the behavioral semantics of metamodels (beyond their basic structure), so that different kinds of analysis can be conducted, e.g., simulation, validation and model checking. A second challenge is the support of the notion of time in these behavioral descriptions, another key issue to allow industrial systems to be realistically simulated and properly analyzed — to be able to conduct, e.g., performance and reliability analysis. Finally, we need not only to tackle the *accidental* complexity involved building software systems, but we should also try to deal with their *essential* complexity. In this sense, the effective use of independent but complementary viewpoints to model large-scale systems, and the specification of correspondences between them to reason about the consistency of the global specifications, is the third of our identified challenges.

2 Adding Behavioral Semantics to DSLs

Domain Specific Languages (DSLs) are usually defined only by their abstract and concrete syntaxes. The abstract syntax of a DSL is normally specified by a metamodel, which describes the concepts of the language, the relationships among them, and the structuring rules that constrain the model elements and their combinations in order to respect the domain rules.

The concrete syntax of a DSL provides a realization of the abstract syntax of a metamodel as a mapping between the metamodel concepts and their textual or graphical representation (see Fig. 1). A language can have several concrete syntaxes. For visual languages, it is necessary to establish links between these concepts and the visual symbols that represent them — as done, e.g, with GMF. Similarly, with textual languages links are required between metamodel elements and the syntactic structures of the textual DSL.

Current DSM approaches have mainly focused on the structural aspects of DSLs. Explicit and formal specification of a model semantics has not received much attention by the DSM community until recently, despite the fact that this creates a possibility for semantic mismatch between design models and modeling languages of analysis tools [1]. While this problem exists in virtually every domain where DSLs are used, it is more common in domains in which behavior needs to be explicitly represented, as it happens in most industrial applications of a certain complexity. This issue is particularly important in safety-critical real-time and embedded system domains, where precision is required and where semantic ambiguities may produce conflicting results across different tools. Furthermore, the lack of explicit behavioral semantics strongly hampers the

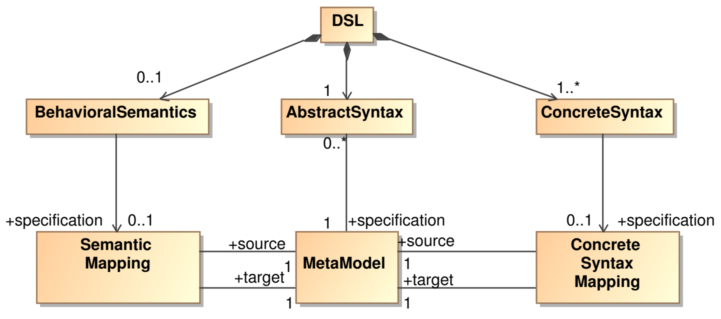


Fig. 1. Specification of a Domain Specific Language

development of formal analysis and simulation tools, relegating models to their current common role of simple illustrations.

The definition of the semantics of a language can be accomplished through the definition of a mapping between the language itself and another language with well-defined semantics (see Fig. 1). These *semantic mappings* [2] are very useful not only to provide precise semantics to DSLs, but also to be able to simulate, analyze or reason about them using the logical and semantical framework available in the target domain. In our opinion, in MDE these mappings can be defined in terms of model transformations.

Describing Dynamic Behavior. There are several ways for specifying the dynamic behavior of a DSL, from textual to graphical. We can find approaches that make use of, e.g., UML diagrams, rewrite logic, action languages or Abstract State Machines [3] for this aim. One particular way is by describing the evolution of the state of the modeled artifacts along some time model. In MDE, model transformation languages that support in-place update [4] can be perfect candidates for the job. These languages are composed of rules that prescribe the preconditions of the actions to be triggered and the effects of such actions.

There are several approaches that propose in-place model transformation to deal with the behavior of a DSL. One of the most important graphical approaches on this topic is graph grammars [5,6], in which the dynamic behavior is specified by using visual rules. These rules are visually specified as models that use the concrete syntax of the DSL. This kind of representation is quite intuitive, because it allows designers to work with domain specific concepts and their concrete syntax for describing the rules [5]. There are also other graphical approaches, most of which are in turn based on graph grammars. Among them, we can find the visual representation of QVT [7] (where QVT is given in-place semantics) or the use of different (usually extended) UML diagrams [8,9]. These approaches do not use (so far) the concrete syntax of the DSL, but an object diagram-like structure. Furthermore, most of them (including graph grammars approaches) use their own textual language to deal with complex behavior, such as Java [8] or Python [10].

Model Simulation and Analysis. Once we have specified the behavior of a DSL, the following step is to perform simulation and analysis over the produced specifications. Defining the model behavior as a model will allow us to transform them into different semantic domains. Of course, not all the transformations can always be accomplished: it depends on the expressiveness of the semantic approach. In fact, simulation and execution possibilities are available for most of the approaches in which behavior can be specified (including of course in-place transformations), but the kind of analysis they provide is normally limited. In general, each semantic domain is more appropriate to represent and reason about certain properties, and to conduct certain kinds of analysis [3].

A good example of this is Graph Transformation, which has been formalized into several semantic domains to achieve different kinds of analysis. Examples include Category theory to detect rule dependencies [11]; Petri Nets to allow termination and confluence analysis [5]; or Maude and rewrite logic to make models amenable to reachability and model-checking analysis [12]. We have been working on the formalization of models and metamodels in equational and rewriting logic using Maude [13]. This has allowed us to specify and implement some of the most common operations on metamodels, such as subtyping or difference [14], with a very acceptable performance. This formalization has also allowed us to add behavior [15] in a very natural way to the Maude specifications, and also made metamodels amenable to other kinds of formal analysis and simulation.

3 Adding Time to Behavioral Specifications

Formal analysis and simulation are critical issues in complex and error-prone applications such as safety-critical real-time and embedded systems. In such kind of systems, timeouts, timing constraints and delays are predominant concepts [16], and thus the notion of time should be explicitly included in the specification of their behavior. Most simulation tools that enable the modeling of time require specialized knowledge and expertise, something that may hinder its usability by the average DSL designer. On the other hand, current in-place transformation techniques do not allow to model the notion of time in a quantitative way, or allow it by adding some kind of clocks to the DSL metamodel. This latter approach forces designers to modify metamodels to include time aspects, and allows them to easily design rules that lead the system to time-inconsistent states [16].

One way to avoid this problem is by extending behavioral rules with their duration, i.e., by assigning to each action the time it needs to be performed. Analysis of this kind of timed rules cannot be easily done using the common theoretical results and tools defined for graph transformations. However, other semantic domains are better suited. We are now working on the definition of a semantic mapping to Real-Time Maude's rewrite logic [17]. This mapping brings several advantages: (1) it allows to perform simulation, reachability and model-checking analysis on the specified real-time systems; (2) it permits decoupling time information from the structural aspects of the DSL (i.e., its metamodel); and (3) it allows to state properties over both model states and actions, easing designers in the modeling of complex systems.

4 Viewpoint Integration and Consistency

Large-scale heterogeneous distributed systems are inherently much more complex to design, specify, develop and maintain than classical, homogeneous, centralized systems. Thus, their complete specifications are so extensive that fully comprehending all their aspects is a difficult task. One way to cope with such complexity is by dividing the design activity according to several areas of concerns, or *viewpoints*, each one focusing on a specific aspect of the system, as described in IEEE Std. 1471. Following this standard, current architectural practices for designing open distributed systems define several distinct viewpoints. Examples include the viewpoints described by the growing plethora of Enterprise Architectural Frameworks (EAF): the Zachman's framework, ArchiMate, DoDAF, TOGAF, FEAF or the RM-ODP. Each viewpoint addresses a particular concern and uses its own specific (viewpoint) *language*, which is defined in terms of the set of concepts specific that concern, their relationships and their well-formed rules.

Although separately specified, developed and maintained to simplify reasoning about the complete system specifications, viewpoints are not completely independent: elements in each viewpoint need to be related to elements in the other viewpoints in order to ensure the *consistency* and *completeness* of the global specifications. 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 their *consistency*. There are many approaches that try to tackle the problem of consistency between viewpoints, many of them coming from the ADL community (see, e.g., [3] for a list of such works). However, many of the current viewpoint modeling approaches to system specification used in industry (including the IEEE Std. 1471 itself and the majority of the existing EAFs) do not address these problems [18].

There are several ways to check viewpoint consistency. In some approaches such as the OpenViews framework [19], two views are consistent if a design can be found that is a refinement of both views. Other approaches, such as Viewpoints [20], consistency requirements are defined in terms of rules, which are specified as queries on the database that contains the viewpoints. The database performs then the consistency checks, using first-order logic. But the most general approach to viewpoint consistency is based on the definition of *correspondences* between viewpoint elements.

Correspondences do not form part of any of the viewpoints, but provide statements that relate the various different viewpoint specifications—expressing their semantic relationships. The problem is that current proposals and EAFs do not consider correspondences between viewpoints, or assume they are trivially based on name equality between correspondent elements, and are implicitly defined. Furthermore, the majority of approaches that deal with viewpoint inconsistencies assume that we can build an underlying metamodel containing all the views, which is not normally true. For instance, should such a metamodel consist of the intersection or of the union of all viewpoints elements? Besides, the granularity and level of abstraction of the viewpoints can be arbitrarily different, and they may have very different semantics, which greatly complicates the definition of the common metamodel.

Our efforts are currently focused on the development of a generic framework and a set of tools to represent viewpoints, views and correspondences, which are able to manage and maintain viewpoint synchronization in evolution scenarios, as reported in [21], and that can be used with the most popular existing EAFs.

References

1. Kleppe, A.G.: A language description is more than a metamodel. In: Proc. of ATEM 2007 (october 2007), `http://megaplanet.org/atem2007/ATEM2007-18.pdf`

2. Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"? Computer 37(10), 64–72 (2004)
3. Vallecillo, A.: A Journey through the Secret Life of Models. In: Position paper at the Dagstuhl seminar on Model Engineering of Complex Systems (MECS) (2008), http://drops.dagstuhl.de/opus/volltexte/2008/1601
4. Czarnecki, K., Helsen, S.: Classification of model transformation approaches. In: OOPSLA 2003 Workshop on Generative Techniques in the context of MDA (2003)
5. de Lara, J., Vangheluwe, H.: Translating model simulators to analysis models. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 77–92. Springer, Heidelberg (2008)
6. Kastenberg, H., Kleppe, A.G., Rensink, A.: Defining object-oriented execution semantics using graph transformations. In: Gorrieri, R., Wehrheim, H. (eds.) FMOODS 2006. LNCS, vol. 4037, pp. 186–201. Springer, Heidelberg (2006)
7. Marković, S., Baar, T.: Semantics of OCL Specified with QVT. Software and Systems Modeling (SoSyM) (2008)
8. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story diagrams: A new graph rewrite language based on the unified modeling language. In: Proc. of the VI International Workshop on Theory and Application of Graph Transformation (1998)
9. Engels, G., Hausmann, J.H., Heckel, R., Sauer, S.: Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In: Evans, A., Kent, S., Selic, B. (eds.) UML 2000. LNCS, vol. 1939, pp. 323–337. Springer, Heidelberg (2000)
10. de Lara, J., Vangheluwe, H.: Defining visual notations and their manipulation through meta-modelling and graph transformation. Journal of Visual Languages and Computing 15(3-4), 309–330 (2006)
11. Ehrig, H., Karsten, Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006)
12. Rivera, J.E., Guerra, E., de Lara, J., Vallecillo, A.: Analyzing rule-based behavioral semantics of visual modeling languages with maude. In: Proc. of SLE 2008, Toulouse, France. LNCS. Springer, Heidelberg (2008)
13. Romero, J.R., Rivera, J.E., Durán, F., Vallecillo, A.: Formal and tool support for model driven engineering with Maude. JOT 6(9), 187–207 (2007)
14. Rivera, J.E., Vallecillo, A.: Representing and operating with model differences. In: Proc. of TOOLS Europe 2008. LNBIP, vol. 11, pp. 141–160. Springer, Heidelberg (2008)
15. Rivera, J.E., Vallecillo, A.: Adding behavioral semantics to models. In: Proc. of EDOC 2007, pp. 169–180. IEEE Computer Society, Los Alamitos (2007)
16. Gyapay, S., Heckel, R., Varró, D.: Graph transformation with time: Causality and logical clocks. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 120–134. Springer, Heidelberg (2002)
17. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20(1-2), 161–196 (2007)
18. Romero, J.R., Vallecillo, A.: Well-formed rules for viewpoint correspondences specification. In: Proc. of WODPEC 2008 (2008)
19. Boiten, E.A., Bowman, H., Derrick, J., Linington, P., Steen, M.W.: Viewpoint consistency in ODP. Computer Networks 34(3), 503–537 (2000)
20. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: a framework for integrating multiple prespectives in systems development. SEKE journal 2(1), 31–58 (1992)
21. Eramo, R., Pierantonio, A., Romero, J.R., Vallecillo, A.: Change management in multi-viewpoint systems using ASP. In: Proc. of WODPEC 2008 (2008)

Embedded System Construction – Evaluation of Model-Driven and Component-Based Development Approaches

Christian Bunse[1], Hans-Gerhard Gross[2], and Christian Peper[3]

[1] International University in Germany, Bruchsal, Germany
Christian.Bunse@i-u.de
[2] Delft University of Technology, Delft, The Netherlands
h.g.gross@tudelft.nl
[3] Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany
Christian.Peper@iese.fraunhofer.de

Abstract. Model-driven development has become an important engineering paradigm. It is said to have many advantages over traditional approaches, such as reuse or quality improvement, also for embedded systems. Along a similar line of argumentation, component-based software engineering is advocated. In order to investigate these claims, the MARMOT method was applied to develop several variants of a small micro-controller-based automotive subsystem. Several key figures, like model size and development effort were measured and compared with figures coming from two mainstream methods: the Unified Process and Agile Development. The analysis reveals that model-driven, component-oriented development performs well and leads to maintainable systems and a higher-than-normal reuse rate.

Keywords: Exploratory Study, Embedded, Model-Driven, Components.

1 Introduction

Embedded software design is a difficult task due to the complexity of the problem domain and the constraints from the target environment. One specific technique that may, at first sight, seem difficult to apply in the embedded domain, is modeling and Model-Driven Development (MDD) with components. It is frequently used in other engineering domains as a way to solve problems at a higher level of abstraction, and to verify design decisions early. Component-oriented development envisions that new software can be created with less effort than in traditional approaches, simply by assembling existing parts. Although, the use of models and components for embedded software systems is still far from being industrial best practice. One reason might be, that the disciplines involved, mechanical-, electronic-, and software engineering, are not in sync, a fact which cannot be attributed to one of these fields alone. Engineers are struggling hard to master the pitfalls of modern, complex embedded systems. What is really lacking is a vehicle to transport the advances in software engineering and component technologies to the embedded world.

Software Reuse is still a challenging area of research. One reason is that software quality and productivity are assumed to be greatly increased by maximizing the (re)use of (part of) prior products, instead of repeatedly developing from scratch. This also stimulated the transfer of MDD and CBD [12] techniques to the domain of embedded systems, but the predicted level of reuse has not yet been reached. A reason might be that empirical studies measuring the obtained reuse rates are sparse. Studies, such as [7] or [8] examined only specific aspects of reuse such as specialization, or off-the-shelf component reuse, but they do not provide comparative metrics on the method's level. Other empirical studies that directly focus on software reuse either address non-CBD technology [14], or they focus on representations on the programming language-level [15]. Unfortunately, there are no studies in the area of MDD/CBD for embedded systems.

This paper shortly introduces the MARMOT system development method. MARMOT stands for *Method for Component-Based Real-Time Object-Oriented Development and Testing*, and it aims to provide the ingredients to master the multi-disciplinary effort of developing embedded systems. It provides templates, models and guidelines for the products describing a system, and how these artifacts are built. The main focus of the paper is on a series of studies in which we compare MARMOT, as being specific for MDD and CBD with the RUP and Agile Development to devise a small control system for an exterior car mirror. In order to verify the characteristics of the three development methods, several aspects such as model size [13] and development effort are quantified and analyzed. The analysis reveals that model-based, component-oriented development performs well and leads to maintainable systems, plus a higher-than-normal reuse rate, at least for the considered application domain.

The paper is structured as follows: Section 2 briefly describes MARMOT, and Sections 3, 4, and 5 present the study, discuss results and address threats to validity. Finally, Section 6 presents a brief summary, conclusions drawn, and future research.

2 MARMOT Overview

Reuse is a key challenge and a major driving force in hardware and software development. Reuse is pushed forward by the growing complexity of systems. This section shortly introduces the MARMOT development method [3] for model-driven and component-based development (CBD) of embedded systems. MARMOT builds on the principles of KobrA [1], assuming its component model displayed in Fig. 1, and extending it towards the development of embedded systems. MARMOT components follow the principles of encapsulation, modularity and unique identity that most component definitions put forward. Component communication relies on interface contracts (i.e., in the embedded world these are made available through software abstractions). An additional hardware wrapper realizes that the hardware communication protocol is translated into a component communication contract. Further, encapsulation requires separating the description of what a software unit does from the description of how it does it. These descriptions are called *specification* and *realization* (see Fig. 1).

The specification is a suite of descriptive (UML [11]) artifacts that collectively define the external interface of a component, so that it can be assembled into, or used by, a system. The realization artifacts collectively define a component's internal realization. Following this principle, each component is described through a suite of models as if it was an independent system in its own right.

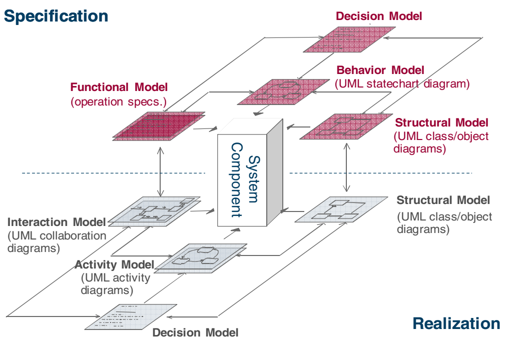


Fig. 1. MARMOT component model

2.1 Process Model

The fact that components can be realized using other components, turns a MARMOT project into a tree-shaped structure with consecutively nested abstract component representations. A system can be viewed as a containment hierarchy of components in which the parent/child relationship represents composition. Any component can be a containment tree in itself, and, as a consequence, another MARMOT project. Acquisition of component services across the tree turns a MARMOT project into a graph. The four basic activities of a MARMOT development process are composition, decomposition, embodiment, and validation as shown in Fig. 2.

Decomposition follows the divide-and-conquer paradigm, and it is performed to subdivide a system into smaller parts that are easier to understand and control. A project always starts above the top left-hand side box in Fig. 2. It represents the entire system to be

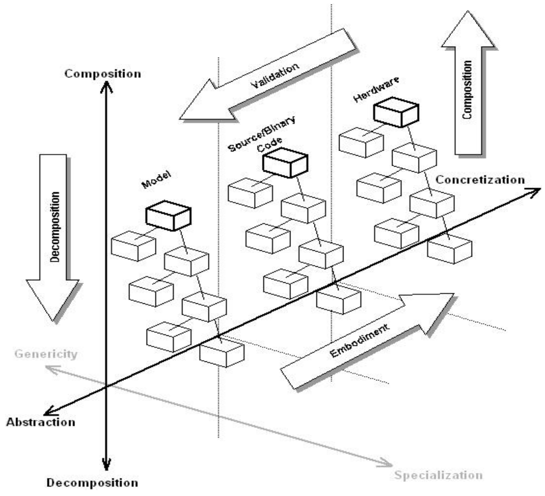


Fig. 2. Development activities in MARMOT

built. Prior to specifying the box, the domain concepts must be determined, comprising descriptions of all relevant domain entities such as standard hardware components that will appear along the concretization dimension. The implementation-specific entities determine the way in which a system is divided into smaller parts. During decomposition, newly identified logical parts are mapped to existing components, or the system is decomposed according to existing components. Whether these are hard- or software is not important since all components are treated in a uniform way: as abstract components.

Composition represents the opposite activity, which is performed when individual components have been implemented or reused, and the system is put together. After having implemented some of the boxes and having some others reused, the system can be assembled according to the abstract model. The subordinate boxes with their respective super-ordinate boxes have to be coordinated in a way that exactly follows the component description standard introduced above.

Embodiment is concerned with the implementation of a system and a move towards executable representations. It turns the abstract system (i.e., models) into concrete representations that can be executed. MARMOT uses refinement and translation patterns for doing these transformations, and it supports the generation of code skeletons. It can, thus, be regarded as a semi-automatic approach.

Validation checks whether the concrete representations are in line with the abstract ones. It is carried out in order to check whether the concrete composition of the embedded system corresponds to its abstract description.

2.2 Product Model

MARMOT follows the principles of encapsulation, modularity **and unique identity,** which lead to a number of obligatory properties**.**

- **Composability** is the primary property and it can be applied recursively: components make up components, which make up components, and so on.
- **Reusability** is the second key property, separated into development for reuse, i.e., components are specified to be reusable, and development with reuse, dealing with the integration and adaptation of existing components in a new application.
- Having **unique identities** requires that a component must be uniquely identifiable within its development and runtime environment.
- **Modularity/encapsulation** refer to a component's scoping property as an assembly of services, which is also true for a hardware component, and as an assembly of common data, which is true for the hardware and the software. The software represents an abstraction of the hardware.

An additional important property is **communication** according to interface contracts which becomes feasible in the embedded world through typical software abstractions. Here, the additional hardware wrapper of MARMOT realizes that the hardware communication protocol is translated into a component communication contract.

Composition turns a MARMOT project into a tree-shaped structure, a containment tree, with nested component representations. Every box in the tree is treated as a system in its own right. It comprises a component specification, defining everything externally knowable about a component, and a component realization, a model about the internal design of the component. Any component in a tree also represents a containment tree, and, thus, another MARMOT project.

3 Description of the Study

In general, empirical studies in software engineering are used to evaluate whether a "new" technique is superior to other techniques concerning a specific problem or property. The objective of this study is to compare the effects of MARMOT concerning reuse in embedded system development to other approaches such as the Unified Process and agile development.

The study was organized in three runs (i.e., one run per methodology). All runs followed the same schema. Based on an existing system documentation, subjects performed a number of small projects. These covered typical project situations such as maintenance, ports to another platform, variant development, and reuse in a larger context. The first run applied MARMOT. The second run repeated all projects but used a variation of the Unified Process, specifically adapted for embedded system development. The third run, applying an agile approach, was used to validate that modeling has a major impact and to rule out that reuse effects can solely be obtained at the code level. Metrics were collected in all runs and were analyzed in order to evaluate the respective research questions.

3.1 Research Approach

Introducing MDD and CBD in an organization is generally a slow process. An organization will start with some reusable components, and eventually build a component repository. But they are unsure about the return on investment gained by initial component development plus reuse for a real system, and the impact of the acquired technologies on quality and time-to-market. This is the motivation for performing the study and asking questions on the performance of these techniques.

Research Questions. Several factors concerning the development process and its resulting product are recorded throughout the study in order to gain insights about using MDD and CBD for the development of small embedded systems. The research questions of the case-study focus on two key sets of properties of MDD in the context of component-oriented development. The first set of questions (Q1-Q4) lead to an understanding of basic and/or general properties of the embedded system development approach:

Q1: Which process was used to develop the system? Each run of the study used a different development approach (i.e., MARMOT, Unified Process, and Agile). These are compared in terms of different quality attributes of the resulting systems.

Q2: Which types of diagrams have been used? Are all UML diagram types required, or is there possibly a specific subset sufficient for this domain?

Q3: How were models transferred to source code? Developers typically work in a procedural setting that impedes the manual transformation of UML concepts into C [10].

Q4: How was reuse applied and organized? Reuse is central to MDD with respect to quality, time-to-market, and effort, but reuse must be built into the process, it does not come as a by-product (i.e., components have to be developed for reuse).

The second set of questions (Q5-Q9) deals with the resulting product of the applied approach (i.e., with respect to code size, defect density, and time-to-market).

Q5: What is the model-size of the systems? MDD is often believed to create a large overhead of models, even for small projects. Within the study, model size follows the metrics as defined in [13].

Q6: What is the defect density of the code?

Q7: How long did it take to develop the systems and how is this effort distributed over the requirements, design, implementation, and test phases? Effort saving is one promise of MDD and CBD [12], though, it does not occur immediately (i.e., in the first project), but in follow-up projects. Effort is measured for all development phases.

Q8: What is the size of the resulting systems? Memory is a sparse resource and program size extremely important. MDD for embedded systems will only be successful if the resulting code size, obtained from the models, is small.

Q9: How much reuse did take place? Reuse is central for MDD and CBD and it must be seen as an upfront investment paying off in many projects. Reuse must be examined between projects and not within one project.

Research Procedure. MDD and CBD promise efficient reuse and short time-to-market, even for embedded systems. Since it is expected that the benefits of MDD and CBD are only visible during follow-up projects [5], an initial system was specified and used as basis for all runs. The follow-ups then were:

R1/R2 Ports to different hardware platforms while keeping functionality constant. Ports were performed within the family (ATMega32) and to a different processor family (PICF). Implementing a port within the same family might be automated at the code-level, whereas, a port to a different family might affect the models.

R3/R4 Evolving system requirements by (1) removing the recall position functionality, and (2) adding a defreeze/defog function with a humidity sensor and a heater.

R5 The mirror system was reused in a door control unit that incorporates the control of the mirror, power windows, and door illumination.

3.2 Preparation

Methodologies. The study examines the effects of three different development methods on software reuse and related quality factors. In the first run, we used the MARMOT method that is intended to provide all the ingredients to master the multidisciplinary effort of developing component-based embedded systems. In the second run we followed an adapted version of the Unified Process for embedded system development [4] (i.e., Rational Unified Process Systems Engineering aka RUP SE). RUP SE includes an architecture model framework that supports different perspectives. A distinguishing characteristic of RUP SE is that the components regarding the perspectives are jointly derived in increasing specificity from the overall system requirements. In the third run, an agile process (based on Extreme Programming) [9], adapted towards embedded software development, was used.

Subjects of the study were graduate students from the Department of Computer Science at the University of Kaiserslautern and the School of IT at the International University. All students, 45 in total (3 per team/project), were enrolled in a Software Engineering class, in which they were taught principles, OO methods, modeling, and embedded system development. Lectures were supplemented by practical sessions in which students had the opportunity to make use of what they had learned. At the beginning of the course, subjects were informed that a series of practical exercises was planned. Subjects knew that data would be collected and that an analysis would be performed, but were unaware of the hypotheses being tested. To further control for learning and fatigue effects and differences between subjects, random assignment to the development teams was performed. As the number of subjects was known before running the studies it was a simple procedure to create teams of equivalent size.

Metrics. All projects were organized according to typical reuse situations in component-based development, and a number of measurements was performed to answer the research questions of the previous sub-section:

Model-size is measured using the absolute and relative size measures proposed in [13]. Relative size measures (i.e., ratios of absolute measures) are used to address UMLs multi-diagram structure and to deal with completeness [13]. Absolute measures used are: the number of classes in a model (NCM), number of components in a model (NCOM), number of diagrams (ND), and LOC, which are sufficient as complexity metrics for the simple components used in this case. NCOM describes the number of hardware/software components, while NCM represents the number of software components. These metrics are comparable to metrics such as McCabe's cyclomatic complexity for estimating the size/nesting of a system. Code-size is measured in normalized LOC. *System size* is measured in KBytes of the binary code. All systems were compiled using size optimization.

The *amount of reused elements* is described as the proportion of the system which can be reused without any changes or with small adaptations (i.e., configuration but no model change). Measures are taken at the model and code level.

Defect density is measured in defects per 100 LOC, whereby defects were collected via inspection and testing activities.

Development effort and its distribution over development phases are measured as development time (hours) collected by daily effort sheets.

Materials. The study uses a car-mirror control (a replication package is available from the authors). The system is composed of electrical and mechanical components and control logic. It allows the mirror to be adjusted horizontally and vertically into the desired position. Cars supporting different driver profiles can store the mirror position and recall as soon as the profile is activated. The system (Fig. 3.) comprises a microcontroller, a button, two potentiometers, and two servos. It controls the two servo-drives via two potentiometers, and indicates their movement on a small LCD panel. The micro-controller reads values from the potentiometers, converts them to degrees, and generates the needed servo control signals, while at the same time indicating movement and degree on the LCD display. The system stores a position through pressing the button for more than 5 seconds. Concerning the Marmot approach the project was executed as follows:

1) *Requirements Modeling*: Use cases describe the requirements in a textual and a graphical representation. Activity diagrams describe the general flow of control, including a UML representation of the target platform.
2) *System Architecture*: The system architecture is described by the system's 'context realization'. The context is like a pseudo component realization at the root of the development tree that embeds the system as a regular component. It specifies how the computer system is affecting the context into which it is embedded. Since components are identified in a top-down manner, a component or containment hierarchy is established.
3) *Component Modeling*: Component modeling creates the specification and realization of all software components using class, state, interaction, and activity diagrams, as well as operation schemata. Since timing is critical in embedded systems, the component realization is extended by timing diagrams. Modelingl

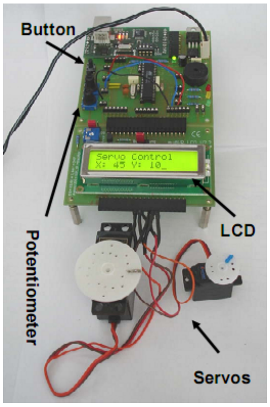


Fig. 3. Exterior mirror control system

starts at the root of the containment hierarchy, and the top-level component is specified using three different UML models: (1) structural model, showing with which other classes the component interacts; (2) functional model, describing the externally visible operations supplied by the component; (3) behavioral model, showing the externally visible state model.

4) The component specification is further decomposed to the component realization comprising the component's private design. It describes how the component fulfills its requirements, via (1) a structural model, showing its internal class architecture, (2) an activity model specifying the algorithms, and (3) an interaction model showing how a group of instances collaborate to realize an operation. These primary artifacts can be enhanced, if needed, by timing diagrams, or other non-functional specifications.
5) *Implementation*: Iteratively devising specifications and realizations is continued until an existing component is found, thereby targeting existing abstractions, or, until it can be implemented (no reuse). Coming to a concrete implementation from the models requires us to reduce the level of abstraction of our descriptions. First, the containment hierarchy is simplified according to the technical restrictions of the used implementation technology. That is through refining the containment hierarchy and mapping it to a UML model with the source code structure of the resulting system. Second, the models are mapped to source code, either through a code generator, or through manual mapping approaches.

For each run, the base system documentation was developed by the authors of this paper. The reason was that we were primarily interested in the reuse effects of one methodology in the context of follow-up projects. Using a single documentation for all runs would have created translation and understanding efforts. Therefore, reasonable effort was spent to make all three documents comparable concerning size, complexity, etc. This is supported by the measures of each system.

4 Evaluation and Comparison

In the context of the three experimental runs, a number of measurements were performed with respect to maintainability, portability, and adaptability of software systems. Tables 1, 2, and 3 provide data concerning model and code size, quality, effort, and reuse rates. Table columns denote the project type[1].

Table 1. Results of the First Run (MARMOT)

		Original	R1	R2	R3	R4	R5
LOC		310	310	320	280	350	490
Model Size (Abs.)	NCM	8	8	8	6	10	10
	NCOM	15	15	15	11	19	29
	ND	46	46	46	33	52	64
Model Size (Rel.)	$\frac{NumberofStateCharts}{NumberofClasses}$	1	1	1	1	0.8	1
	$\frac{NumberofOperations}{NumberofClasses}$	3.25	3.25	3.25	2.5	3	3.4
	$\frac{NumberofAssociations}{NumberofClasses}$	1.375	1.375	1.375	1.33	1.3	1.6
Reuse	Reuse Fraction(%)	0	100	97	100	89	60
	New (%)	100	0	3	0	11	40
	Unchanged (%)	0	95	86	75	90	95
	Changed (%)	0	5	14	5	10	5
	Removed (%)	0	0	0	20	0	40
Effort (h)	Global	26	6	10.5	3	10	24
	Hardware	10	2	4	0.5	2	8
	Requirements	1	0	0	0.5	1	2
	Design	9.5	0.5	1	0.5	5	6
	Implementation	3	1	3	0.5	2	4
	Test	2.5	2.5	2.5	1	2	4
Quality	Defect Density	9	0	2	0	3	4

First Run. Porting the system (R1) required only minimal changes to the models. One reason is that MARMOT supports the idea of platform-independent modeling (platform specific models are created in the embodiment step). Ports to different processor families (R2) are supported by MARMOT's reuse mechanisms.

Concerning the adaptation of existing systems (R3 and R4), data show that large portions of the system could be reused. In comparison to the initial development project the effort for adaptations is quite low (26 hrs vs. 3 or 10 hrs). The quality of the system profits from the quality assurance activities of the initial project. Thus, the promises of CBD concerning time-to-market and quality could be confirmed.

[1] Project types are labeled following the scheme introduced in section 3 (e.g., "Original" stands for the initial system developed by the authors as a basis for all follow-up projects, "R1" – Port to the ATMEGA32 microcontroller (same processor family), "R2" – Port to the PIC F microcontroller (different processor family), "R3" – Adaptation by removing functionality from the original system, "R4" – Adaptation by adding functionality to the original system, and "R5" – Reuse of the original system in the context of a larger system.

Table 2. Results of the Second Run (Unified Process)

		Original	R1	R2	R3	R4	R5
LOC		350	340	340	320	400	500
Model Size (Abs.)	NCM	10	10	10	8	12	13
	NCOM	15	15	15	11	19	29
	ND	59	59	59	45	60	68
Model Size (Rel.)	$\frac{NumberofStateCharts}{NumberofClasses}$	1.5	1.5	1.5	0.72	1.33	1.07
	$\frac{NumberofOperations}{NumberofClasses}$	4	3.5	3.5	3.25	3	3.46
	$\frac{NumberofAssociations}{NumberofClasses}$	2.5	2.3	2.3	2.5	2.16	1.76
Reuse	Reuse Fraction(%)	0	100	94	88	86	58
	New (%)	100	0	6	11	14	42
	Unchanged (%)	0	92	80	70	85	86
	Changed (%)	0	4	15	6	15	14
	Removed (%)	0	4	5	24	0	41
Effort (h)	Global	34	8	12	5.5	13	29
	Hardware	10	2	4	0.5	2	8
	Requirements	4	1	1	1.5	3	4
	Design	12	1	2	1	4	7
	Implementation	5	2	3	1.5	2	6
	Test	3	2	2	1	2	4
Quality	Defect Density	8	1	2	0	3	4

Interestingly, the effort for the original system corresponds to standardized effort distributions over development phases, whereby the effort of follow-ups is significantly lower. This supports the assumption that component-oriented development has an effort-saving effect in subsequent projects.

Porting and adapting an existing system (R1-R4) implies that the resulting variants are highly similar, which explains why reuse works well. It is, therefore, interesting to look at larger systems that reuse (components of) the original system (i.e., R5). 60% of the R5 system was reused without requiring major adaptations of the reused system. Effort and defect density are higher than those of R1-R4, due to additional functionality and hardware extensions. However, when directly compared to the initial effort and quality, a positive trend can be seen that supports the assumption that MARMOT allows embedded systems development at a low cost but with high quality.

The **Second and Third Run** replicated the projects of the first run but used different development methods. Interestingly, the results of the second run are quite close to those of the first. However, the Unified Process requires more overhead and increased documentation, resulting in higher development effort. Ironically, model-size seems to have a negative impact on quality and effort. Interestingly, the mapping of models to code seems not to have added additional defects or significant overheads.

Although the amount of modeling is limited in the agile approach, it can be observed that the original system was quickly developed with a high quality. However, this does not hold for follow-up projects. These required substantially higher effort than the effort required for runs 1 and 2. A reason might be that follow-ups were not performed by the developers of the original system. Due to missing documentation and abstractions, reuse rates are low. In contrast, the source-code appears to be of good quality.

Table 3. Results of the Third Run (Agile)

		Original	R1	R2	R3	*R4*	R5
LOC		280	290	340	300	*330*	550
Model Size (Abs.)	NCM	14	15	15	13	*17*	26
	NCOM	5	5	5	4	*7*	12
	ND	3	3	3	3	*3*	3
Model Size (Rel.)	$\frac{NumberofStateCharts}{NumberofClasses}$	0	0	0	0	*0*	0
	$\frac{NumberofOperations}{NumberofClasses}$	3.21	3.3	3.3	3.15	*3.23*	4.19
	$\frac{NumberofAssociations}{NumberofClasses}$	3.5	3.3	3.3	3.46	*3.17*	2.57
Reuse	Reuse Fraction(%)	0	95	93	93	*45*	25
	New (%)	100	5	7	7	*55*	75
	Unchanged (%)	0	85	75	40	*54*	85
	Changed (%)	0	14	15	40	*36*	10
	Removed (%)	0	1	10	20	*10*	5
Effort (h)	Global	18	5	11.5	6	*13.5*	37
	Hardware	6	2	4	1	*2*	8
	Requirements	0.5	0	0	0.5	*1*	1
	Design	2	0	0	1	*1.5*	3
	Implementation	7	2	5	2	*6*	18
	Test	2.5	1	2.5	1.5	*3*	7
Quality	Defect Density	7	0	2	1	*5*	7

5 Threats to Validity

The authors view this study as exploratory. Thus, threats limit generalization of this research, but do not prevent the results from being used in further studies.

Construct Validity. Reuse is a difficult concept to measure. In the context of this paper it is argued that the defined metrics are intuitively reasonable measures. Of course, there are several other dimensions of each concept. However, in a single controlled study it is unlikely that all the different dimensions of a concept can be captured.

Internal Validity. A maturation effect is caused by subjects learning as the study proceeds. The threat to this study is subjects learned enough from single runs to bias their performance in the following ones. An instrumentation effect may result from differences in the materials which may have caused differences in the results. This threat was addressed by keeping the differences to those caused by the applied method. This is supported by the data points as presented in table 1, 2, and 3. Another threat might be the fact that the studies were conducted at different institutes.

External Validity. The subjects were students and unlikely to be representative of software professionals. However, the results can be useful in an industrial context for the following reasons: Industrial employees often do not have more experience than students when it comes to applying MDD. Furthermore, laboratory settings allow the investigation of a larger number of hypotheses at a lower cost, than in field studies. Hypotheses supported in the laboratory setting can be tested further in industrial settings.

6 Summary and Conclusions

The growing interest in the Unified Modeling Language provides a unique opportunity to increase the amount of modeling work in software development, and to elevate quality standards. UML 2.x promises new ways to apply object/component-oriented and model-based development techniques in embedded systems engineering. However, this chance will be lost, if developers are not given effective and practical means for handling the complexity of such systems, and guidelines for applying them systematically.

This paper shortly introduced the MARMOT approach that supports the component-oriented and model-based development of embedded software systems. A series of studies was described that were defined to empirically validate the effects of MARMOT on aspects such as reuse or quality in comparison to the Unified Process and an agile approach. The results indicate that using MDD and CBD for embedded system development will have a positive impact on reuse, effort, and quality. However, similar to product-line engineering projects, CBD requires an upfront investment. Therefore, all results have to be viewed as initial. This has led to the planning of a larger controlled experiment to obtain more objective data.

References

[1] Atkinson, C., Bayer, J., Bunse, C., et al.: Component-Based Product-Line Engineering with UML. Addison-Wesley, Reading (2001)

[2] Bunse, C., Gross, H.-G., Peper, C.: Applying a Model-based Approach for Embedded System Development. In: 33rd SEAA, Lübeck, Germany (2007)

[3] Bunse, C., Gross, H.-G.: Unifying hardware and software components for embedded system development. In: Reussner, R., Stafford, J.A., Szyperski, C. (eds.) Architecting Systems with Trustworthy Components. LNCS, vol. 3938, pp. 120–136. Springer, Heidelberg (2006)

[4] Cantor, M.: Rational Unified Process for Systems Engineering, the Rational Edge e-Zine (2003), `http://www.therationaledge.com/content/aug_03/f_rupse_mc.jsp`

[5] Crnkovic, I., Larsson, M. (eds.): Building Reliable Component-Based Software Systems. Artech House (2002)

[6] Douglass, B.P.: Real-Time Design Patterns. Addison-Wesley, Reading (2003)

[7] Briand, L.C., Bunse, C., Daly, J.W.: A Controlled Experiment for Evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs. IEEE TSE 27(6) (2001)

[8] Li, J., Conradi, R., Slyngstad, O.P.N., Torchiano, M., Morisio, M., Bunse, C.: A State-of-the-Practice Survey of Risk Management in Development with Off-the-Shelf Software. IEEE Transaction on Software Engineering 34(2) (2008)

[9] Hruschka, P., Rupp, C.: Agile SW-Entwicklung für Embedded Real-Time Systems mit UML, Hanser (2002)

[10] Marwedel, P.: Embedded System Design (Updated Version). Springer, Heidelberg (2006)

[11] Object Management Group, UML Infrastructure and Superstructure, V2.1.2 (2007)

[12] Szyperski, J.: Component Software. Beyond OOP. Addison-Wesley, Reading (2002)

[13] Lange, C.F.: Model Size Matters. In: Workshop on Model Size Metrics, 2006 (co-located with the ACM/IEEE MoDELS/UML Conference) (October 2006)

[14] Burkhard, J.-M., Detienne, F.: An Empirical Study of Software Reuse By Experts in Object-Oriented Design. In: INTERACT 1995, Lillehammer Norway, June 27-29 (1995)

[15] Lee, N.-Y., Litecky, C.R.: An Empirical Study of Software Reuse with Special Attention to ADA. IEEE Transaction on Software Engineering 23(9) (1997)

Assessing the Power of a Visual Modeling Notation – Preliminary Contemplations on Designing a Test –

Dominik Stein and Stefan Hanenberg

Universität Duisburg-Essen
{dominik.stein,stefan.hanenberg}@icb.uni-due.de

Abstract. This paper reports on preliminary thoughts which have been conducted in designing an empirical experiment to assess the comprehensibility of a visual notation in comparison to a textual notation. The paper sketches shortly how a corresponding hypothesis could be developed. Furthermore, it presents several recommendations that aim at the reduction of confounding effects. It is believed that these recommendations are applicable to other experiments in the domain of MDE, too. Finally, the paper reports on initial experiences that have been made while formulating test questions.

1 Introduction

Although modeling does not imply visualization, people often consider the visual representation of models to be a key characteristic of modeling. One reason to this could be that modeling techniques such as State Machines or Petri-Nets are often taught and explained with help of circles and arrows rather than in terms of mathematical sets and functions. Apart from that, other kinds of modeling, e.g. data modeling with help of Entity-Relationship-Diagrams, make heavy use of visual representations, although the same concepts could be specified in a purely textual manner, too.

However, let alone the impression that visual representations are considered very appealing by a broad range of developers, customers, maintainers, students, etc., a scientific question would be if a visual representation for a particular purpose actual yields any extra benefit to software development, maintenance, teaching, etc. (cf. [11, 12, 13, 16, 17, 18, 19]).

Driven by a personal belief of the authors that this extra benefit exists, this paper reports on preliminary thoughts which have been conducted in designing an empirical experiment. The goal of this empirical experiment is to assess (such a "soft" property as) the "comprehensibility" of a visual notation in comparison to a textual notation.

This (workshop) paper does not formulate a concrete hypothesis, nor does it present any results. Instead, it conducts general contemplation about the design of an empirical test comparing the comprehensibility of a visual and a textual notation. In particular, the paper presents several recommendations that aim at the reduction of confounding effects while running the test. It is suggested that these recommendations should be obeyed in other experiments evaluating visual notations in the domain of MDE, too. Furthermore, the paper reports on experiences that have been made while formulating the test questions for a test on comprehensibility.

The paper is structured as follows: Section 2 outlines the general goal of the experiment and what to measure. Sections 3 discusses proper design of the test objects. Section 4 discusses proper treatment of the test subjects (i.e. the testers). Section 5 deals with proper measurement of the outcomes, and reports on problems which have been encountered during a preliminary pilot test. Section 6 presents related work, and section 7 concludes the paper.

2 Defining the Goal of the Experiment, and What to Measure?

This section is meant to "set the scene" for the subsequent sections on confounding impacts (sections 3 and 4) and preliminary experiences (section 5) with/on the comparison of a visual and a textual notation. To do so, we give a (rough) definition of the experiment, sketch a hypothesis, and select the variables we want to measure.

2.1 Experiment Definition

An experiment comparing visual vs. textual notations can be defined as follows (using the experiment definition template suggested by [31]):

The goal of the study is to analyze visual and textual program specifications (i.e. diagrams versus code), with the purpose of evaluating their effect on the "comprehensibility" of the information shown. The quality focus is the perception speed and completeness and correctness with which all relevant information is apprehended. The perspective is that of teachers and program managers, who would like to know the benefit that visual notations can bring to their work (i.e. teaching students in computer science or developing software). The context of the experiment is made up of artificial/sample code snippets and their corresponding diagrams (= objects) as well as undergraduate and graduate students (= subjects).

2.2 Hypothesis Formulation

According to [22], a scientific hypothesis meets the following three criteria:

- A hypothesis must be a "for-all" statement (or rather, a "for-all-meeting-certain-criteria" statement). This means in particular that the hypothesis must be true for more than a singular entity or situation.
- A hypothesis must be (able to be reformulated as) a conditional clause (of the form "whenever A is true/false, this means that B is (also) true/false").
- A hypothesis must be falsifiable. That means that, in principle, it must be able to find an entity or situation in which the hypothesis is *not* true.

Furthermore, for practical reasons, [4, 23] suggest to base the hypothesis on observable data. That is, in the (possibly reformulated) conditional clause, the value of one observable data (called "the dependent variable") must be specified to depend on the value of one other observable data (called "the independent variable") in a consistent way. The hypothesis is falsified if at least one example can be found where this dependency is not satisfied.

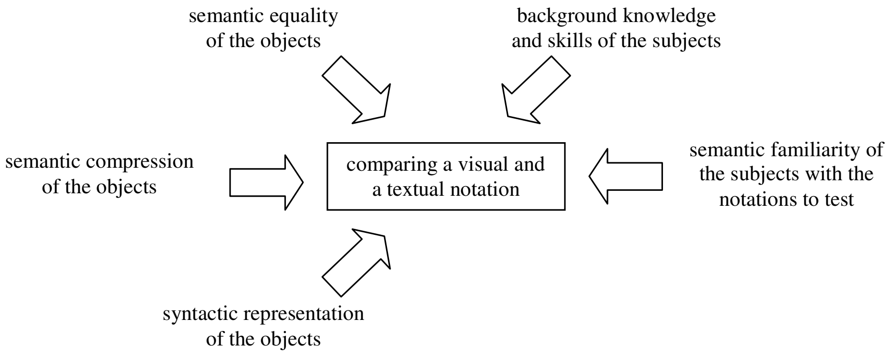

semantic equality of the objects
background knowledge and skills of the subjects
semantic compression of the objects
comparing a visual and a textual notation
semantic familiarity of the subjects with the notations to test
syntactic representation of the objects

Fig. 1. Confounding factors

A starting point to find a hypothesis for the experiment outlined in section 2.1 could be the following:

When investigating program specifications, a visual representation X (as compared to a textual representation Y) ***significantly facilitates*** *comprehension of information Z.*

Following the aforementioned criteria, the above hypothesis is a scientific hypothesis because it can be rephrased as "whenever a program specification is represented using a visual notation X, it is easier to comprehend (with respect to information Z) than an equivalent representation using a textual notation Y". In this statement, the possible values (treatments) of the independent variable (factor) are "visual/not visual" and the possible values of the dependent variable are "easier to comprehend/not easier to comprehend". The claimed dependency would be "visual $\rightarrow$ easier to comprehend". The statement could be falsified by showing that visual notation X is not easier to comprehend than textual notation Y (with respect to information Z).

2.3 Variable Selection

Turning a the preliminary idea of a hypothesis into a testable hypothesis which is thoroughly rooted on objectively observable data is a challenging task in developing an empirical test. Comprehensibility by itself, for example, is difficult to observe. Hence, another variable must be found whose values are considered to depend on the level of comprehension of a tester. A commonly accepted variable measuring the level of comprehension, for example, is "correctness", i.e. the number of correct answers given to a (test) questions (cf. [29, 28, 27]). However, as pointed out by [28], correctness is only one facet of comprehensibility. Another variable is "comprehension speed", e.g. the number of seconds that the subjects look at the object (or maybe even "easy to remember", i.e. the number of times that the subjects take a look at the objects; cf. [28]). The inherent effect of the variable being of interest on the variable being measured must be substantially elucidated (and defended) in the discussion on the (construct) validity of the test.

The only factor (independent variable) in the experiment would be "kind of presentation" with the treatments (values) {visual, textual}.

One of the big challenges when investigating the casual dependencies between the (dependent and independent) variables is to reduce confounding impacts (see Fig. **1**) as much as possible, and thus to maximize the validity of the experiment (cf. [31]). Otherwise, the "true" dependency could possibly be neutralized (at least, in parts), or might even be turned into its reciprocal direction (in the worst case).

In the following sections, some means are presented which should be taken in order to improve the validity of an experiment comparing a visual and a textual notation. The authors believe that these means are general enough to be applied to other experiments evaluating two notations in the domain of MDE, too.

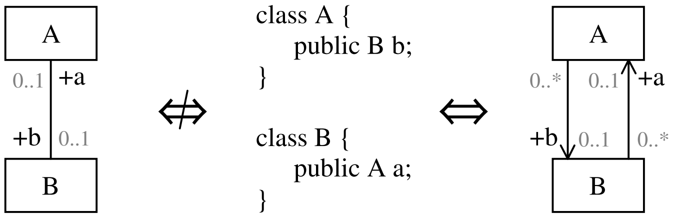
3 Preparing Objects – Ensuring Construct Validity (I)

Construct validity refers "to the extent to which the experiment setting actually reflects the construct under study" [31]. In particular, this means to ensure that the objects of the experiment which are given to the subjects in order to perform the tests represent the cause well (i.e. a visual vs. a textual representation, in this case).

3.1 Semantic Equality

One obvious, yet carefully to ensure, requirement is to compare (visual and textual) representations that have equal semantics, only. It would be illegal and meaningless to compare any two representations with different semantics.

Ensuring semantic equality is a greater challenge, though, than it might appear at the first glance. Fig. 2 shows a (simple) example.



```
class A {
    public B b;
}

class B {
    public A a;
}
```

Fig. 2. Ensure semantic equality

The bidirectional association between classes `A` and `B` in the UML model in the left of Fig. 2 denotes that two instances of class `A` and `B` are related to each other such that the instance of class `A` can navigate to the instance of class `B` via property `b`, while at the same time the instance of class `B` can navigate to the instance of class `A` via property `a` (meaning `a = a.b.a` is always true; cf. [20, 9, 1, 10]). The Java program code in the middle of Fig. 2, however, does not imply that an instance of class `A` which is associated with an instance of class `B` (via its property `b`) is the same instance which that associated instance of class `B` can navigate to via its property `a` (meaning `a = a.b.a` does not need to be true).

Therefore, in an empirical test comparing the performance[1] of visual vs. textual representations of associations, it would be meaningless to compare the textual representation in the middle of Fig. 2 with the visual representation in the *left* of Fig. 2 since they mean different things. Instead, it would be more appropriate to compare the textual representation in the middle of Fig. 2 with the visual representation in the *right* of Fig. 2. Here, the meaning of one representation can be considered equal to the meaning of the other representation (cf. [9]), and comparing the results of their individual performance can be considered valid.

It must be noted that, in general, asserting the semantic equality of two notations is not trivial. This is particularly true if the semantics of the two notations have not been specified in compatible formalisms. In the case of UML and Java, for example, this means that there is no general agreement on how a UML class diagram should be transformed into Java code. There is a considerable amount of work that suggests possible transformations (e.g. [9, 1, 10]). These approaches make use of additional methods (e.g. setter and getter methods), or even entire classes, to properly capture the (more advanced parts of the) semantics of UML associations in Java. In doing so, these approaches ensure the semantic equality of visual and textual representations considerably well, thus improving the validity of the experiment. However, at the same time, they threaten the validity by conflicting with another requirement, which is discussed next: the requirement of equal degree of semantic compression.

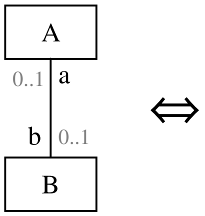
3.2 Equal Degree of Compression

Apart from semantic equality, the expressions being compared need to be expressed at an equal degree of compression (here, the degree of compression shall refer to the degree with which semantic information is condensed into one language construct; the use of the term "compression" in this context – rather than "abstraction" – is inspired by [8, 14]). Otherwise, "better" performance of one notation could be induced by the fact that one notation uses a "higher" compression (e.g. one language construct of that notation conveys the same semantic information than four language constructs of the other notation) rather than that it uses a "better" representation.

Fig. 3 gives an example. Other than in Fig. 2, the Java code now contains extra lines which states that an instance of class `A` which is associated with an instance of class `B` (via its property `b`) must be the same instance to which that associated instance of class `B` can navigate via its property `a` (meaning `a = a.b.a` is always true; see [1] for a more sophisticated solution). Hence, the Java expression in the right of Fig. 3 now equates to the semantics of the UML expression in the left of Fig. 3.

If – in a test – the UML expression should actually yield "better" results than the Java expression now, it is unclear (and highly disputable) whether the "better" performance is due to the visual representation or due to the higher degree of compression (i.e. the fact that we need to read and understand four method definitions in the Java code as compared to just one association in the UML diagram). A comparison of two, i.e. a visual and a textual, notations with different compression is, of course, still valid and may show that one notation performs "better" than the other. However, it

[1] In this paper, "performance" refers to "the notation's ability to be read and understood" rather than computation speed.



```
class A {
    B b;
    B getB() { return b; }
    void setB(B b) { this.b = b; b.a = this; }
}

class B {
    A a;
    A getA() { return a; }
    void setA(A a) { this.a = a; a.b = this; }
}
```

Fig. 3. Do not test expressions of unequal degree of compression

cannot be argued that this is an immediate consequence from the one being visual, while the other is not.

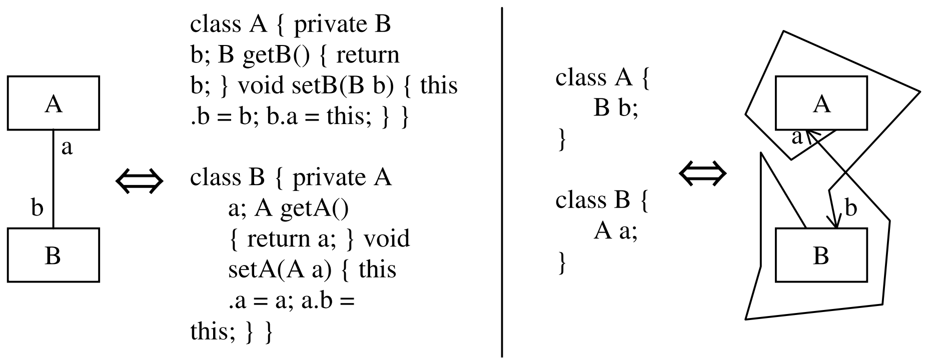
3.3 Presenting Objects

Apart from equal semantics and equal degree of compression, the expressions have to be appropriately formatted, each to its cleanest and clearest extent. This is because the authors estimate that disadvantageous formatting of expressions could have a negative impact on the test outcome, whereas advantageous formatting of expressions could improve the test results. There is an extensive body of research supporting this assumption (e.g. [18, 21, 24, 25, 3, 17] to name just a few).

Fig. 4 gives an example. In the left part of Fig. 4, the Java code has been formatted in a way which is tedious to read. In the right part of Fig. 4, the UML representation has been formatted disadvantageously. With expressions formatted like this, it is assumed that the respective notation is condemned to fail in the performance test.

Unfortunately, there usually is no (known) optimal solution for the formatting task. Therefore, expressions should be formatted clearly and consistently following some strict and predefined guidelines (e.g. according to [30, 2, 7]). Furthermore, syntactic sugar is to be avoided. That is, all means that are not related to the semantics of the underlying notation, such as syntax highlighting in textual expressions, or different text formats and different line widths in visual expressions, should not be used. Syntactic sugar (fonts, colors, line width, etc.) is likely to impact the comprehensibility of the expressions (cf. [26, 3] for studies) and thus may confound the pure comparison between their visual and textual representation.

Evaluating the impacts of formatting, fonts, indentation, colors, and line width on the comprehensibility of a notation is an interesting test of its own. However, that test should focus on the comparison of different style guidelines for *one* notation rather than on the comparison of (different) guidelines for different notations. Several experiments have indicated that even slight variations in the representation may impact the performance of a (visual or textual) notation [18, 24, 25, 3, 17, 26]. Hence, it is important to keep in mind that even though uniform guidelines are used to format the expressions, the effects of those formatting guidelines on the test outcomes are



```
class A { private B
b; B getB() { return
b; } void setB(B b) { this
.b = b; b.a = this; } }

class B { private A
    a; A getA()
    { return a; } void
    setA(A a) { this
    .a = a; a.b =
this; } }
```

```
class A {
    B b;
}

class B {
    A a;
}
```

Fig. 4. Format expressions to their cleanest and clearest extent

unclear. This is all the more true because the (unknown) effects may be different for each notation! Consequently, the impact of formatting guidelines on the test results needs to be respected in the discussion of the (construct) validity of the test.

4 Preparing Subjects – Ensuring Internal Validity

To ensure internal validity, it must be ensured that a relationship between a treatment and an outcome results from a causal relationship between those two, rather than from a factor which has not been controlled or has not been measured (cf. [31]). In particular this means how to "treat", select, and distribute the subjects such that no coincidental unbalance exists between one group of testers and another.

4.1 Semantic Familiarity

The imperative necessity of comparing semantically equivalent "expressions" (see section 3.1) is complemented with the necessity that testers are equally trained in, and familiar with, both notations. Otherwise, i.e. if the testers of one notations are more experienced with their notation than the testers of the other notation with their notation, a "better" test result of the former notation could be induced by the fact that its testers have greater experience in using/reading it rather than by the fact that it is actually "better" (in whatsoever way). This effect has become evident in several experiments (cf. [5, 6, 21]) – where [6] points out that this is particularly true in case of briefly presented programs, which may be common in many empirical tests.

The overall challenge here is to determine under which circumstances two testers can be considered "equally familiar" (i.e. equally knowing and skilled) with their notations. Furthermore, it needs to be investigated how the knowledge and skills of an individual tester (with his/her notation) can be reliably assessed (so that we can decide afterwards whether or not "equal familiarity" has been reached). Finally, it must be considered how "equal familiarity" can be achieved in a timely and didactically appropriate manner (e.g., by a teaching course or brief introduction just prior to the test; yet, what is to be done if a particular group of testers encounters unforeseen comprehension problems with their notation and, thus, spends much more time with their notation than the other group with the other notation?).

5 Measuring Outcomes – Ensuring Construct Validity (II)

Once the hypothesis is sufficiently clear, the next challenging step is to formulate questions that are suitable to test the hypothesis and to find a test format that is suitable to poll the required data. This is another facet of construct validity, according to which the outcome of the test needs to represent the effects well (cf. [31]).

In this section, considerations and experiences are presented that have been made in formulating test questions and capturing answers in/for a pilot test evaluating the comprehensibility of a (textual) notation[2].

5.1 Test Format, and How to Measure?

Multiple Choice tests (when carefully designed; cf. [15]) are considered to be a good and reliable way to test the knowledge of a person, in particularly in comparison to simple True/False tests. Hence, Multiple Choice tests would have a higher construct validity with respect to the correctness of comprehension than True/False tests. A question format with free answer capabilities would be more realistic (and thus would increase the external validity of the experiment; cf. [31]). However, such short-answer test is much more laborious because it requires manual post-processing in order to detect typos and/or semantically equivalent answers.

When it comes to measuring the response time, it is important to discriminate between the time to find the answer in the expression and the time to understand the question. This is because if testers need 30 sec. to understand a question and then 10 sec. to find the answer in the textual expression and just 5 sec. to find the answer in the visual expression, it makes a difference whether 40 sec. are compared to 35 sec., or 10 sec. to 5 sec. Not to discriminate between the time to find an answer and the time to understand a question is only valid, if the ratio is reciprocal, i.e. if the time to understand a question is negligible short in comparison to the time to find the answer.

If the test outcome consists of more than one data, it is a big challenge to define how the outcomes can be combined in order to obtain a meaningful interpretation. In the case discussed here, for example, it needs to be decided how "correctness of answers" and "response time" can be combined to indicate a "level of comprehension". One option would be to disregard all incorrect answers, and consider the response time of correct answers, only.

5.2 Volatile (Time) Measurements – Problems of a First Test Run

Preliminary and repeated test runs of a test evaluating simple analysis of a textual notation[2] (with the same person) have shown that the measured time needed to answer the question (exclusive of the time needed to understand the question; cf. section 5.1) is rather short (in average ~10 sec.) and varies tremendously (3 sec. to 30+ sec., even for same questions; cf. Fig. 5). Against all expectations, no learning effect is evident, which should have yielded similar performance and (at least, slight) improvements after each test run. It seems as if the measured time is heavily confounded by some

[2] In another case than association relationships.

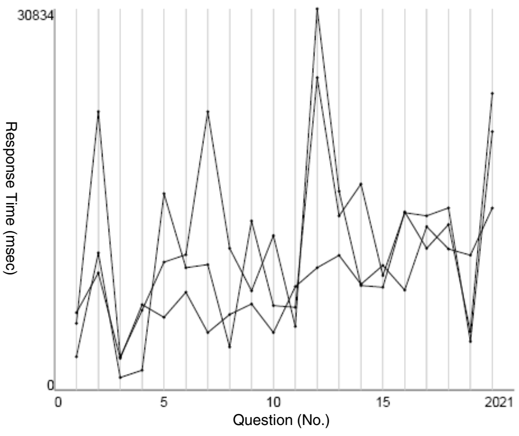


Fig. 5. Preliminary pilot test: Three subsequent test runs with same person, same objects, and same questions (in arbitrary order), leading to highly volatile outcomes (measurements of one test run are connected by a line).

external factor (maybe slight losses of concentration). This is problematic because due to the short (average) response time, even the slightest disturbance (of about 1 sec.) could confound the measured (average) time significantly (e.g. by one tenth, in this case).

Another problem was to strictly discriminate between the time to find the answer in the expression and the time to understand the question (which, again, was essential due to the short (averaged) response time). The testers were required to explicitly flip to the expression once they have carefully read (and understood) the question (which was shown first). As it turned out, however, testers sometimes realized that they have not fully understood the question after they have already flipped to the expression. As a result, the measured response time was partly confounded.

It is currently being investigated how the problem of high variation in measurements can be tackled. One option would be to pose questions that are more difficult to answer, and thus takes more time. This will only work, though, if the confounding effects do not grow proportionally. Another option would be to repeat the test countless times (with the same person and similar questions) in order to get a more reliable average response time. A big problem of this approach is to ensure that the testers will not benefit from learning effects in the repeated tests (which are expected to take effect, ultimately, i.e. some time after the third test run).

A promising solution to properly discriminate between the time to find the answer in the expression and the time to understand the question has been found in [28].

6 Existing Comparisons of Visual and Textual Notations

In 1977, Shneiderman et al. [29] have conducted a small empirical experiment that tested the capabilities of flow charts with respect to comprehensibility, error detection, and modification in comparison to pseudo-code. Their outcome was that – statistically – the benefits of flow charts was not significant. Shneiderman et al. did not measure time, though.

This was determined to be inevitable by Scanlan [28]. Scanlan formulated five hypotheses (e.g. "structured flow charts are faster to comprehend", "structured flow charts reduce misconceptions", to name just the two which are closely related to this paper). Scanlan's test design is very interesting: Scanlan separated comprehension (and response) time of the question from comprehension time of the expression. To do so, testers could *either* look at the question *or* look at the expression (an algorithm, in this case). This is an interesting solution for the aforementioned problem of separating comprehension time and response time (see section 5.1). Scalan's outcome was that structured flow charts are beneficial.

Response time was also considered relevant in subsequent experiments evaluating the performance of visual and textual representations, e.g. [13, 19, 12]. [13] compares a visually enhanced textual representation (called "Control Structure Diagrams") with a plain textual representation of code, and concludes that the visually enhancements have a positive impact on program comprehension tasks. [19] compares a visual and a textual representation of algebraic specifications (called "OBJ" and "VisualOBJ"). In this case, the overall results, although slightly favoring the visual notations, show no statistically significant difference. [12] compares visual notations for conditional logic with corresponding textual notations. The outcome is that visual notations perform worse than textual notations.

All of the aforementioned experiments [29, 28, 13, 19, 12] compare (visual and textual) expressions which are both semantically equivalent (cf. section 3.1) and of equal compression (cf. section 3.2). It is important to note, though, that all of these experiments deal with (visual representations of) low-level programming language constructs. Hence, ensuring semantic equivalence and equal compression is no issue here because there is a one-to-one mapping between visual and textual constructs. In the dawn of MDE, however, the authors of this paper anticipate the development of (visual) domain-specific languages which map to a conglomeration of programming language constructs. This is the situation where empirical experiments evaluating the performance of the visual (i.e. domain-specific) and textual (i.e. code) representations need to respect the confounding impact that unequal compression may have on the measured outcomes.

Apart from comparing visual and textual notations, the experiment in [12] also verifies a previous finding (cf. [11, 16]), i.e. that the performance of a given notation depends on the actual comprehension task which is to be performed. The experiments [11, 12] show, in particular, that an "if-then-else" notation facilitates the answering of so-called "forward questions" (i.e. "which action results into a given condition?"), while a "do-if" notation facilitates answering so-called "backward questions" (i.e. "which condition must hold to trigger a specified action?"). These findings emphasize that the actual comprehension task that shall be studied (i.e. "information Z"; cf. section 2.2) must be chosen just as carefully as the notations that shall be studied. (i.e. "visual representation X" and "textual representation Y"; cf. section 2.2).

7 Conclusion

This paper has presented preliminary thoughts which have been conducted in designing an empirical experiment to assess the comprehensibility of a visual notation in comparison to a textual notation. In particular, the paper has presented several recommendations that aim at the reduction of confounding effects on the measured data. These recommendations are considered helpful for other experiments evaluating two notations in the domain of MDE, too. Furthermore, the paper has reported on initial experiences that have been made while formulating sensible test questions for a preliminary pilot test.

It needs to be emphasized that this paper presents preliminary considerations rather than sustainable outcomes. On the contrary, each of the presented contemplations could be subject of an empirical evaluation of itself (e.g. whether or not semantic compression really has an effect on comprehensibility). Apart from that, decisions need to be made about how to execute the test (e.g. how textual and visual expressions are shown to the testers, if they can use zooming or layouting functions, etc.). The authors plan to pursue the considerations presented here in order to come up with a test design, which is qualified to lead to sustainable test results.

Acknowledgement

The authors thank the anonymous reviewers for their patients with the tentativeness of these contemplations and for their productive comments which have helped to further advance the test design. Furthermore, the authors thank the workshop organizers and workshop participants – in particular, Bente Anda, Erik Arisholm, Lionel Briand, Michel Chaudron, Marcela Genero, Susanne Patig, and Jon Whittle – for their valuable critics, questions, hints, and suggestions which have helped to complement this publication.

References

1. Akehurst, D., Howells, G., McDonal-Maier, K.: Implementing Associations: UML 2.0 to Java 5. Software and Systems Modeling (SoSyM) 6(1), 3–35 (2007)
2. Ambler, S.W.: The Elements of UML 2.0 Style. Cambridge University Press, Cambridge (2005)
3. Andriyevska, O., Dragan, N., Simoes, B., Maletic, J.I.: Evaluating UML Class Diagram Layout based on Architectural Importance. In: Proc. of VISSOFT 2005, pp. 14–19. IEEE, Los Alamitos (2005)
4. Bortz, J., Döring, N.: Forschungsmethoden und Evaluation für Sozialwissenschaftler (Research Methods and Evaluation for Social Scientist). Springer, Heidelberg (1995)
5. Burkhardt, J.-M., Détienne, F., Wiedenbeck, S.: Object-Oriented Program Comprehension: Effect of Expertise, Task and Phase. Empirical Software Engineering 7(2), 115–156 (2002)
6. Davies, S.P.: Expertise and the Comprehension of Object-Oriented Programs. In: Proc. of Workshop of the Psychology of Programming Interest Group, pp. 61–66 (2000)
7. Eichelberger, H.: Nice Class Diagrams Admit Good Design? In: Proc. of SoftVis 2003, pp. 159–167. ACM, New York (2003)
8. Gabriel, R.: Patterns of Software - Tales from the Software Community. Oxford University Press, Oxford (1996)
9. Génova, G., del Castillo, C.R., Llorens, J.: Mapping UML Associations into Java Code. Journal of Object Technology (JOT) 2(5), 135–162 (2003)
10. Gessenharter, D.: Mapping the UML2 Semantics of Associations to a Java Code Generation Model. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 813–827. Springer, Heidelberg (2008)

11. Green, T.: Conditional Program Statements and their Comprehensibility to Professional Programmers. Journal of Occupational Psychology 50(2), 93–109 (1977)
12. Green, T., Petre, M., Bellamy, R.K.E.: Comprehensibility of Visual and Textual Programs: A Test of Superlativism Against the 'Match-Mismatch' Conjecture. In: Empirical Studies of Programmers: 4th Workshop, pp. 121–146 (1991)
13. Hendrix, T.D., Cross II, J.H., Maghsoodloo, S.: The Effectiveness of Control Structure Diagrams in Source Code Comprehension Activities. IEEE Trans. Software Eng. 28(5), 463–477 (2002)
14. Henney, K.: Overload 45 (October 2001), `http://accu.org/index.php/journals/432`
15. Krebs, R.: Die wichtigsten Regeln zum Verfassen guter Multiple-Choice Fragen (Most Important Rules for Writing Good Multiple-Choice Questions), IAWF, Berlin (1997)
16. McGuinness, C.: Problem Representation: The Effects of Spatial Arrays. Memory & Cognition 14(3), 270–280 (1986)
17. Miara, R.J., Musselman, J.A., Navarro, J.A., Shneiderman, B.: Program Indentation and Comprehensibility. Comm. of the ACM 26(11), 861–867 (1983)
18. Moher, T.G., Mak, D.C., Blumenthal, B., Leventhal, L.M.: Comparing the Comprehensibility of Textual and Graphical Programs: The Case of Petri-Nets. In: Empirical Studies of Programmers: 5th Workshop, pp. 137–161 (1993)
19. Neary, D.S., Woodward, M.R.: An Experiment to Compare the Comprehensibility of Textual and Visual Forms of Algebraic Specifications. Journal of Visual Languages and Computing 13(2), 149–175
20. Object Management Group (OMG), UML 2.1.1 Superstructure Specification, Document formal/2007-02-05
21. Petre, M.: Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. Commun. of the ACM 38(6), 33–44 (1995)
22. Popper, K.: Logik der Forschung (The Logic of Scientific Discovery, 1959) (1934)
23. Prechelt, L.: Kontrollierte Experimente in der Softwaretechnik (Controlled Experiments in Software Engineering). Springer, Heidelberg (2001)
24. Purchase, H.C., Colpoys, L., McGill, M., Carrington, D., Britton, C.: UML Class Diagram Syntax: An Empirical Study of Comprehension. In: Proc. of Australasian Symposium on Information Visualisation, pp. 113–120 (2001)
25. Purchase, H.C., Colpoys, L., McGill, M., Carrington, D.: UML Collaboration Diagram Syntax: An Empirical Study of Comprehension. In: Proc. of VISSOFT 2002, pp. 13–22. IEEE, Los Alamitos (2002)
26. Rambally, G.K.: The Influence of Color on Program Readability and Comprehensibility. In: Proc. of SIGCSE 1986, pp. 173–181. ACM, New York (1986)
27. Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., Ceccato, M.: The Role of Experience and Ability in Comprehension Tasks supported by UML Stereotypes. In: Proc. of ICSE 2007, pp. 375–384. IEEE, Los Alamitos (2007)
28. Scanlan, D.A.: Structured Flowcharts Outperform Pseudocode: An Experimental Comparison. IEEE Software 6(5), 28–36 (1989)
29. Shneiderman, B., Mayer, R., McKay, D., Heller, P.: Experimental Investigations of the Utility of Detailed Flowcharts in Programming. Comm. of the ACM 20(6), 373–381 (1977)
30. Sun, Code Conventions for the Java Programming Language, April 20 (1999), `http://java.sun.com/docs/codeconv/`
31. Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., Wesslen, A.: Experimentation in Software Engineering - An Introduction. Kluwer, Dordrecht (2000)

Third International Workshop on Models@run.time

Nelly Bencomo[1], Gordon Blair[1], Robert France[2], Freddy Muñoz[3], and Cédric Jeanneret[4]

[1] Computing Department, Lancaster University, InfoLab21, Lancaster, UK
{nelly,gordon}comp.lancs.ac.uk
[2] Computer Science Department, Colorado State University Fort Collins, CO, USA
france@cs.colostate.edu
[3] IRISA, INRIA, Equipe Triskell, Rennes, France
fmunoz@irisa.fr
[4] Department of Informatics, University of Zurich, Zurich, Switzerland
jeanneret@ifi.uzh.ch

Abstract. The third edition of the workshop Models@run.time was held at the ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MODELS). The workshop took place in the beautiful city of Toulouse, France, on the 30th of October, 2008. The workshop was organised by Nelly Bencomo, Robert France, Gordon Blair, Freddy Muñoz, and Cédric Jeanneret. It was attended by at least 44 people from more than 10 countries. In this summary we present an overview of the presentations and fruitful discussions that took place during the 3rd edition of the workshop Models@run.time.

Keywords: model-driven engineering, reflection, runtime adaptation.

1 Introduction

This year's workshop aimed to build upon the insights gained at workshops held in 2006 and 2007 to better understand the relationship between models produced during development and models used to support and enable runtime monitoring, adaptation and evolution of software. The workshop successfully brought together researchers from different communities. At least forty-four (44) people attended from: Canada, Colombia, France, Germany, Ireland, Israel, Norway, Spain, Switzerland, UK, and the US.

This is the third in a series of MODELS workshops. Therefore, we wanted to take advantage of the experience gained at the two previous editions and focus the discussions of this workshop on the topic: "*from abstract concepts to concrete realizations*".

We aimed to provide a forum for exchange and collaboration among researchers from different communities, including researchers working on model-driven software engineering, software architectures, computational reflection, adaptive systems, autonomic and self-healing systems, and requirements engineering. Thus, the workshop covered a wide range of topics, including relevance and suitability of different model-driven approaches to monitoring and managing systems during runtime, compatibility (or tension) between different model-driven approaches, the role of reflection in maintaining the causal connection between models and runtime systems, experience related to the use of runtime models to adapt software systems, and the use of models to validate and verify behaviour at runtime.

In response to the call for papers, twenty (20) papers were submitted, of which six (6) papers were accepted. Additionally, six (6) short papers were invited for short presentations and a demo was also presented. Each submitted paper was reviewed by at least 3 program committee members. After discussions, two papers were selected as the best papers. The decision was not easy and took into account the relevance of the papers to the workshop theme, the impact on the workshop discussions and outcomes, and the quality of the papers and presentations. We also held a poll of participants to determine their views on which papers were the most noteworthy. The authors of these two papers have now extended and improved their manuscripts taking into account the discussions of the workshop. The extended papers are published in this proceedings.

2 Workshop Format

The workshop was designed to primarily facilitate focused discussion on the use of models during run time. It was structured into presentations, discussion sessions, and a panel. The opening presentation was given by *Nelly Bencomo*. *Nelly* set the context of the workshop reminding the audience of the general goal, and presenting some results from the last two editions of the workshop in MoDELS'06 and MODELS'07. She also described the specific goals of the third edition of the workshop "*from abstract concepts to concrete realizations*" and presented the path to follow during the rest of the day.

After the opening presentation, the paper sessions followed. There were 6 long and 6 short presentations divided in two sessions during the morning. During the afternoon a demo that supports the use of models@run.time was presented, followed by discussion sessions. A panel consisting of three experienced researchers in the area and three representatives from each discussion group discussed how current visions of runtime models can be realized and exploited in practice.

During the presentation session, authors presented their papers. Long presentations were limited to twenty minutes, including five minutes for questions and discussion. Short presentations were limited to five minutes. Presentation sessions were co-chaired by *Øystein Haugen* and *Robert France*. At the end of the presentation session, research interests and questions were discussed. This discussion led to the formation of three breakout groups charged with carrying out more focused discussions during the afternoon.

The panel was chaired by *Gordon Blair* and included *Bran Selic, Øystein Haugen*, and *Jean-Marc Jézéquel* who prepared their presentations in advance. The other three members of the panel were chosen by their colleagues during discussion groups. The workshop was closed by a final discussion session, including an evaluation of the workshop made by the attendees. Details of the sessions and panel are provided in Section 4.

3 Session Summaries

The 6 long and 6 short presentations were divided into the following two categories according to their topics and contributions:

Session 1: Specific Techniques for Models@run.time

Long papers

- *Runtime Models for Self-Adaptation in the Ambient Assisted Living Domain*, Daniel Schneider and Martin Becker.
- *FAME---A Polyglot Library for Metamodeling at Runtime*, Adrian Kuhn and Toon Verwaest.
- *Modeling and Validating Dynamic Adaptation*, Franck Fleurey, Vegard Dehlen, Nelly Bencomo, Brice Morin, and Jean-Marc Jézéquel.

Short papers

- *A Runtime Model for Monitoring Software Adaptation Safety and its Concretisation as a Service*, Audrey Occello, Anne-Marie Dery-Pinna, and Michel Riveill.
- *Runtime Models to Support User-Centric Communication*, Yingbo Wang, Peter J. Clarke, Yali Wu, Andrew Allen, and Yi Deng.
- *An Execution Platform for Extensible Runtime Models*, Mario Sanchez, Ivan Barrero, Jorge Villalobos, and Dirk Deridder

Session 2: Architecture and Frameworks for Models@run.time

Long papers

- *Embedding State Machine Models in Object-Oriented Source Code*, Michael Striewe, Moritz Balz, and Michael Goedicke.
- *Model-Based Traces*, Shahar Maoz.
- *Mutual Dynamic Adaptation of Models and Service Enactment in ALIVE*, Athanasios Staikopoulos, Sebastien Saudrais, Siobhan Clarke, Julian Padget, Owen Cliffe, and Marina De Vos.

Short papers

- *A Framework for bridging the gap between design and runtime debugging of component-based applications*, Guillaume Waignier, Prawee Sriplakich, Anne-Francoise Le Meur, and Laurence Duchien.
- *A Model-Driven Approach for Developing Self-Adaptive Pervasive Systems*, Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano .

- *Model-driven Management of Complex Systems*, Brian Pickering, Sylvain Robert, Stephane Menoret, and Erhan Mengusoglu.

A demo illustrating the use of models at runtime opened the afternoon session:

- *K@RT: An Aspect-Oriented and Model-Oriented Framework for Dynamic Software Product Lines*, Brice Morin, Olivier Barais and Jean-Marc Jézéquel.

Following this demonstration, discussions groups were established. Each group received the same questions to discuss. These questions were based on the specific theme of the workshop for that day, "*from abstract concepts to concrete realizations*":

- *Are we ready to make an impact (assessment of state of the art/ promising ideas/ gaps)?*
- *What are the next steps (how to make this impact/ from abstract concept to concrete realization).*

4 Discussions and Panel

After the presentations, the participants were organized into three groups that met in the afternoon. After spending some time discussing the presentations and shared research interests, the groups came back to the meeting room to present a summary of their discussions and positions. The summaries were presented in a panel by panellists representing the groups. The representatives the discussion groups were *Frank Fleurey, Peter J. Clarke*, and *Stéphane Ménoret* who joined *Bran Selic, Øystein Haugen*, and *Jean-Marc Jézéquel.*

The panel started with *Bran Selic* presenting his position. He defined a "runtime model" as a model that is required in the course of software execution. Similar to a design-time model, a runtime model supports reasoning about a complex system, and can assist in the automated generation of implementations. However, in addition, a runtime model supports dynamic state monitoring and control of complex systems during execution, and supports semantic (re-)integration of possibly heterogeneous software elements at runtime (e.g. through the use of dynamically adaptable metamodels).

According to Bran, the role of these runtime models implies some form of automated treatment that involves access to, interpretation, and generation of the model. Bran suggested the following research challenges:

- Develop methods and standards for specifying semantics suited to automated interpretation.
- Achieve reversible model transformations, to deal with synchronization issues.
- Provide support for dynamic model synthesis, for runtime adaptation (e.g. a tool which builds a model of its users as they use it so that it can adapt its user interface to their habits).
- Develop standardized reflective interfaces.

- Discover mechanism for dynamic reclassification of models, which shall bring into light some patterns and methods of modelling or analyzing models.
- Build model execution engines which can execute specification models.

Bran added that these automated treatments must be efficient and responsive to the changing demands of the running system.

Øystein Haugen continued the panel presenting his views on models@run.time by attempting to provide answers to the questions what is it?, what was it?, and what might it be? *Øystein* also talked about his experience working with several software systems such as SIMULA runtime libraries, runtime modelling in the Specification and Description Language (SDL) for a Train Control system (done in the early nineties), and UML model based on state machines an their relevance to the theme of the workshop. *Øystein* also argued that there is no reason why a modeller should be forced to think in terms of a specific programming language, like Java, rather than in UML terms or any domain specific language (DSL) when dealing with a program that is being executed. He supported his claim while running an explaining a demo of a runtime model (in UML2) based on state machine models. The demo showed how runtime models can be used to visualize the position of mobile devices on a map. The adaptive software updates *GoogleEarth* images according to the position of the mobile devices.

Frank Fleurey, as the representative of his discussion group, noted last year's edition was devoted to the definition of models@run.time and their possible usage and that papers of the workshop edition this year showed that some progress has been made in these directions. In some of these papers, an application is instrumented and feedback is presented to the users at the model level. It has been shown how models@run.time can be used to monitor, validate or debug an application, and to support dynamic adaptation and evolution of software.

Frank's discussion group agreed that there are several ideas and technologies available to support models@run,time in practice. For example, programming languages include introspection or reflection mechanisms and component frameworks provide adaptation mechanisms. All these elements present a great potential to support the use of models@run.time and many papers presented at this workshop leverage them.

However, one of the papers ("FAME---A Polyglot Library for Meta-modelling at Runtime" by Adrian Kuhn and Toon Verwaest) proposed a modelling framework dedicated to models@run.time and thus casted some doubt on the adequacy of current MDE techniques. If a new modelling formalism must indeed be developed for models@run.time, then practical realizations of models@run.time will be inevitably delayed until appropriate modelling formalism is developed.

Peter Clarke continued the panel presenting the position of his discussion group. *Peter* talked about how maintaining a model of the system at runtime can support the adaptation of a system during execution. Indeed, the runtime model of the system potentially provides an abstraction of the running system allows the administrator or any other monitoring systems to determine properties of the running system and take some action to heal, adapt or evaluate the system.

There was an agreement in his group that one promising use of models@runtime is for answering dynamic "what if" questions during execution. At runtime, models potentially allow the user to “play” with the model before a change is made to the running system. Also, models@runtime may be more effective for those systems that are domain specific or user-centric. Such systems tend to focus on a smaller application space and can be represented at a level of abstraction that can benefit the particular stakeholder. Peter, emphasized that for this to happen, the visualization of models@runtime must be improved so a domain specialist can effectively analyze it.

Stéphane Ménoret represented a discussion group in this panel. According to his group, models@run.time is a research area that requires more attention from industry making reference to the position given by Bran Selic who is also from industry. His discussion group also considered important the maintenance of requirement models during execution to check during execution how requirements agree with the capabilities of the “current” system . A similar initiative is supported by Anthony Finkelstein (requirement reflection) [1]. He also stressed the importance of an international distributed lab initiative focused on models@run.time where different people from different research areas in academia and industry could collaborate together.

For *Jean-Marc Jézéquel*, the notion of models@runtime is an idea that has been (implicitly) around at least fifty years and that was already used implicitly in Simula and more recently with Java, and also with UML/matlab/simulink as presented by Ostein. For example, the class-object pattern can be seen as a model that allows the modification of behaviour. What is new is the possibility to make models evolvable. Jean-Marc sees models@runtime as the intersection of computational reflexion and models and make possible to explore dynamic "what if" situations, to decide whether or not to take a given path during execution. To illustrate his position, *Jean-Marc* gave the example of a fire-fighter (in this case the hypothetical running system) in a room suddenly breaking ablaze, with huge flames and rapidly raising temperature. The fire-fighter builds quickly a mental model of his situation and his options. Once he finds an escape that he estimates safe enough, he runs for it.

Final Remarks at the end of the workshop

A general wrap-up discussion was held at the very end of the workshop. The organizers asked for anonymous written feedback about the selection of the best two papers to publish in this proceeding.

The workshop was closed with a warm “thank you” from the organizers to all participants for another successful workshop. We regret that this time the big number of attendees in the workshop did not allow the organization of the usual dinner we have after the workshop. Instead, attendees dispersed to choose from the many good culinary options that Toulouse offers.

After the workshop

After the workshop and conference, more work was needed. Organizers used the feedback from attendees and program committee members to select the best two papers. After discussion the following papers were selected as the best two papers and are published in new versions in this proceeding:

[1] Requirements Reflection a short talk presented by Anthony Finkelstein at the Dagstuhl workshop on self-adaptive systems (January, 2008).

- *Model-Based Traces* by Shahar Maoz.
- *Modeling and Validating Dynamic Adaptation* by Franck Fleurey, Vegard Dehlen, Nelly Bencomo, Brice Morin, Jean-Marc Jézéquel.

A survey was prepared after the workshop and 24 people answered to this survey. People confirmed that they were pleased with the discussions carried out during the workshop and considered them useful to their own research. They also appreciated the introduction of the panel in the format of the workshop. From the survey and comments at the end of the workshop in Toulouse, it was agreed that the topic models@run.time is relevant for the MODELS community and that this community should be encouraged to continue the study the issues related to the topic.

Acknowledgments. No workshop is successful by the efforts of only a few people. We would also like to thank the members of the program committee who acted as anonymous reviewers and provided valuable feedback to the authors: *Betty Cheng, Fabio Costa, Jeff Gray, Oystein Haugen, Jozef Hooman, Gang Huang, Paola Inverardi, P.F.Linington, Jean-Marc Jézéquel, Rui Silva Moreira, Andrey Nechypurenko, Oscar Nierstrasz, Eugenio Scalise, Arnor Solberg, Thaís Vasconcelos Batista*, and *Steffen Zschaler*. We specially thank to *Bran Selic, Oystein Haugen, Jean-Marc Jézéquel, Peter Clarke, Frank Fleurey,* and *Stéphane Ménoret* for the feedback and ideas provided for this summary. Last but not least, the authors for their interesting submissions are thanked for helping us making this workshop possible.

Modeling and Validating Dynamic Adaptation[*]

Franck Fleurey[1], Vegard Dehlen[1], Nelly Bencomo[2], Brice Morin[3], and Jean-Marc Jézéquel[3]

[1] SINTEF, Oslo, Norway
[2] Computing Department, Lancaster University, Lancaster, UK
[3] IRISA/INRIA Rennes, Equipe Triskell, Rennes, France

Abstract. This paper discusses preliminary work on modeling and validation dynamic adaptation. The proposed approach is on the use of aspect-oriented modeling (AOM) and models at runtime. Our approach covers design and runtime phases. At design-time, a base model and different variant architecture models are designed and the adaptation model is built. Crucially, the adaptation model includes invariant properties and constraints that allow the validation of the adaptation rules before execution. During runtime, the adaptation model is processed to produce a correct system configuration that can be executed.

1 Introduction

In [10], we presented our work on how we combine model-driven and aspect-oriented techniques to better cope with the complexities during the construction and execution of adaptive systems, and in particular on how we handle the problem of exponential growth of the number of possible configurations of the system. The use of these techniques allows us to use high-level domain abstractions and simplify the representation of variants. The fundamental aim is to tame the combinatorial explosion of the number of possible configurations of the system and the artifacts needed to handle these configurations. We use the notion of models at runtime [2] to generate the adaptation logic by comparing the current configuration of the system and a newly composed model that represent the configuration we want to reach. One of the main advantages is that the adaptation does not have to be manually specified.The adaptation model covers the adaptation rules that drive the execution of the system. These rules can be dynamically introduced to change the behavior of the system during execution. We also discussed in [10] the need of techniques to validate the adaptation rules at design-time. In this paper we discuss our preliminary work on how to perform simulation and allow for model-checking in order to validate adaptation rules at design-time. The model validated at design-time is used at runtime.

The remainder of this paper is organized as follows. Section 2 presents an overview of our methodology for managing dynamic adaptation. Section 3 gives details on our meta-model for adaptive systems, and shows through a service discovery example how it can be used to model variability, context, adaptation rules and constraints.

[*] This work is done in the context of the European collaborative project DiVA (Dynamic Variability in complex, Adaptive systems).

Section 4 shows how we simulate the adaptation model to validate the adaptation rules. Section 5 explains our solution for runtime model-based adaptation. Finally, Section 6 discusses the main challenges our work is facing and concludes.

2 Overview of the Approach

Figure 1 presents the conceptual model of the proposed approach. From a methodological perspective the approach is divided in two phases: design-time and runtime.

At design-time, the application base and variant architecture models are designed and the adaptation model is built. At runtime, the adaptation model is processed to produce the system configuration to be used during execution. The following paragraphs details the steps of Figure 1.

Since the potential number of configurations for an adaptive system grows exponentially with the number of variation points, a main objective of the approach is to model adaptive systems without having to enumerate all their possible configurations statically. In order to achieve this objective, an application is modeled using a *base model* which contains the common functionalities and a set of *variant models* which can be composed with the base model. The variant models capture the variability of the adaptive application. The actual configurations of the application are built at runtime by selecting and composing the appropriate variants. The adaptation model does not deal with the basic functionality which is represented by the base model. Instead, the adaptation model just deals with the adaptive parts of the system represented by the variant models. The adaptation model specifies which variants should be selected according to the adaptation rules and the current context of the executing system.

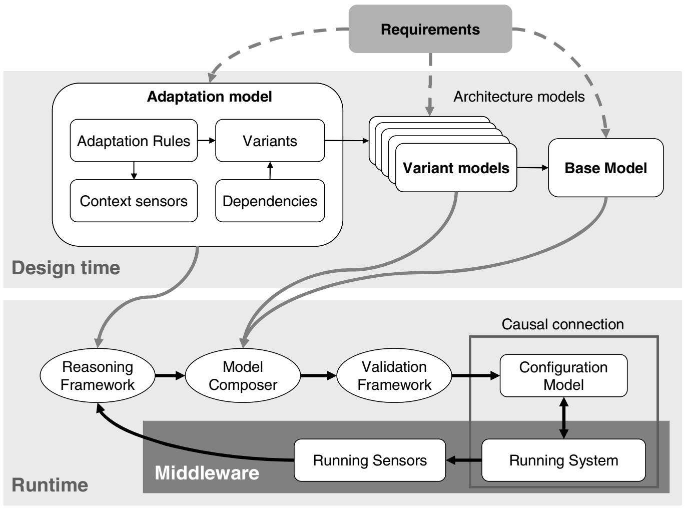


Fig. 1. Overview of the proposed approach

The adaptation model is central to the approach as it captures all the information about the dynamic variability and adaptation of the adaptive system. It is built from the requirements of the system, refined during design and used at runtime to manage adaptation. The adaptation model has four main elements:

- *Variants*: They make references to the available variability for the application. Depending on the complexity of the system, it can be a simple list of variants, a data structure like a hierarchy, or a complex feature model.
- *Constraints*: They specify constraints on variants to be used over a configuration. For example, the use of a particular functionality (variant model) might require or exclude others. These constraints reduce the total number of configurations by rejecting invalid configurations.
- *Context*: The context model is a minimal representation of the environment of the adaptive application to support the definition of adaptation rules. We only consider elements of the environment relevant for expressing adaptation rules. These elements are updated by sensors deployed on the running system.
- *Rules*: These rules specify how the system should adapt to its environment. In practice these rules are relations between the values provided by the sensors and the variants that should be used.

During runtime appropriate configurations of the application are composed from the base and variant models. In order to select the appropriate configuration, the reasoning framework processes the adaptation model and makes a decision based on the current context. The output of the reasoning framework is a configuration that matches the adaptation rules and satisfies the dependency constraints. The model of this configuration can be built at runtime using model composition.

3 Adaptation Model

This section presents the adaptation meta-model and how it is applied to a Service Discovery Application (SDA). The SDA is a solution to tackle heterogeneity of service discovery protocols are presented in [7]. The solution allows an application to adapt to different service discovery protocols and needs during execution. The service discovery platform can take different roles that individual protocols could assume:

-User Agent (*UA*) to discover services on behalf of clients,
-Service Agent (*SA*) to advertise services, and,
-Directory Agent (*DA*) to support a service directory.

Depending on the required functionality, participating nodes might be required to support 1, 2, or the 3 roles at any time. A second variability dimension is the specific service discovery protocols to use, such as ALLIA, GSD, SSD, SLP [7]. Each service discovery protocol follows its own rules. As a result, in order to get two different agents understanding each other, they need to use the same protocol [10]. These decisions have to be performed during execution.

The next sub-section presents an overview of the meta-model and the following sub-sections detail how it is instantiated for the SDA example.

3.1 Meta-model for Variability and Adaptation

As detailed in the previous section the adaptation model includes four different aspects: *variants*, *adaptation rules*, *dependencies* and *context*. Additionally, links to the architecture models and concepts for rules and expressions are supplied. The meta-model is shown in Figure 2. As can be seen from the figure, colors are used to differentiate between the categories.

The colors indicate the following:

- *Grey* – base and aspect architecture models;
- *Orange* – variability information;
- *Purple* – adaptation rules;
- *Red/pink* – dependencies, formulated as constraints;
- *Yellow* – context information;
- *Blue* – expressions.

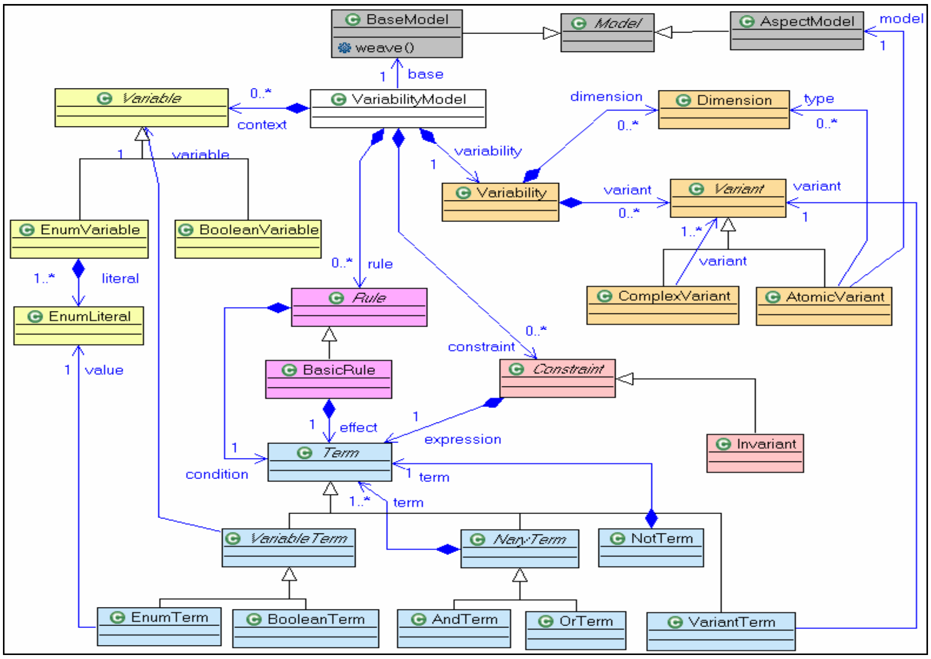


Fig. 2. Meta-model for variability and adaptation

This is to be considered a first version of our meta-model that has been created at an early stage in the DiVA project. It was created based on a set of simple examples such as the SDA described in this paper. During the project, the meta-model will evolve based on feedback and experiences with applying it to larger and more complex case studies. Nevertheless, at this point the meta-model is able to support modeling, simulation and validation activities. The following shows how the meta-model is instantiated for the SDA. To make the example readable we use a textual concrete syntax. This concrete syntax is processed by our prototype tool in order to build the adaptation model.

3.2 Modeling Variability

Figure 3 shows a model of the variability information in our service discovery example, located in the section identified by the *#variability k*eyword. We start by defining two variability dimensions: one for functional variability and another for different discovery protocols that the application can use. A variability dimension can best be described as a category of variants, while a variant is an aspect or concern that is described outside of the base model and may vary to produce adaptation. So far, we have specialized variants into atomic variants and complex variants. The latter is used to express a collection of several variants, thus forming a partial or full configuration. This concept was added because we encountered in our example that some combinations of variants can be foreseen during the requirements phase. As an example, the Discovery Agent functionality corresponds to having both the User Agent and the Service Agent functionalities. DA is thus defined as a complex variant referring to UA and SA.

```
#variability /* Variability of the application */

dimension Functionality : UA, SA
variant DA : UA, SA

dimension DiscoveryProtocol : ALLIA, SLP
```

Fig. 3. Variability in the Service Discovery Application

3.3 Modeling the Context

Information about the context and sensors are delimited by the *#context* keyword. Currently, the meta-model supports two types of context variables: Booleans and enumerations.

The context model, as shown in Figure 4, starts with defining a variable for whether or not the device is running low on battery and, similarly, if the application has been elected as a Discovery Agent. Next, we have defined an enumeration that holds different roles.

```
#context /* Context of the system */

boolean LowBatt // Battery is low
// Node has been elected Discovery Agent
boolean ElectedDA

// Node is required to act either as
// User Agent or as Service Agent
enum SrvReq : UA, SA

// Node is require to use one or
// more of the following prototcols
boolean ALLIAReq
boolean SLPReq
```

Fig. 4. Context of the Service Discovery Application

The application has to act as one of these roles at all time. Finally, there are two variables that tell which protocols are required, which can be one or many.

3.4 Modeling Adaptation

Once the variability and context have been modeled, the adaptation rules can be specified. The adaptation rules link the context variables and the variants in order to specify the configuration to use with respect to a particular context. Currently, adaptation is based on simple *condition-action* rules. The *condition* part is a Boolean expression based on the context information, while the *action* is a change in the configuration of variants.

```
/* Adaptation rules for functionalities */

rule BecomeDA : // Becomes a DA
  condition ElectedDA and not LowBatt and not DA
  effect DA

rule StopDA : // Stop being a DA
  condition (LowBatt or not ElectedDA) and DA
  effect not DA

rule BecomeUA : // Become a User Agent
  condition  SrvReq=UA and not UA
  effect UA and not SA

rule BecomeSA : // Become a Service Agent
  condition SrvReq=SA and not SA
  effect not UA and SA
```

Fig. 5. Adaptation rules for the functionalities of the SDA

Figure 5 depicts the adaptation rules for the variants in the functionality category. The first rule is called "*BecomeDA*", which is triggered when an application is elected as a discovery agent. If the device also has sufficient batteries and it is not a discovery agent already, the adaptation will proceed and the application will assume the role of a discovery agent.

3.5 Modeling Constraints

Finally, Figure 6 shows the dependencies. These are currently modeled as constraints, more specifically invariants. For example, the first invariant states that the application must use at least one functionality variant. If it does not, an error message will be produced by the tool.

```
invariant AtLeastOneFunctionality : UA or SA
invariant NotDAWithLowBatt : not (LowBatt and DA)
invariant AtLeastOneProtocol : ALLIA or SLP
invariant NoSLPWithLowBatt : not (SLP and LowBatt)
```

Fig. 6. Invariants of the SDA

4 Simulation and Validation

The main benefit of using a model to describe adaptation is that it enables to process this model at design-time in order to validate it [13]. Based on the meta-model defined in the previous section we have defined a simulator and automated the verification of invariants. This section describes the way the simulator is built and how it allows checking for termination of adaptation rules and verification of invariant properties.

4.1 Simulation Model and Implementation

The goal of the simulation is to build a model of the valid potential configurations and adaptations of the application. To do that, the simulation starts from an initial configuration and applies the adaptation rules to move to a new configuration. Figure 7 presents the simulation model. According to this model, a simulation is composed of a set of configurations and a set of adaptations between these configurations. Each configuration refers to a set of variants and a set of variable terms. The variants correspond to the aspect to be woven in order to build this configuration [7]. The Variable terms define the state of the context variables for this configuration. An adaptation links a source configuration with a target configuration. An adaptation is triggered by a context event and refers to one or more adaptation rules. The context event is a change in the values of one or more context variables.

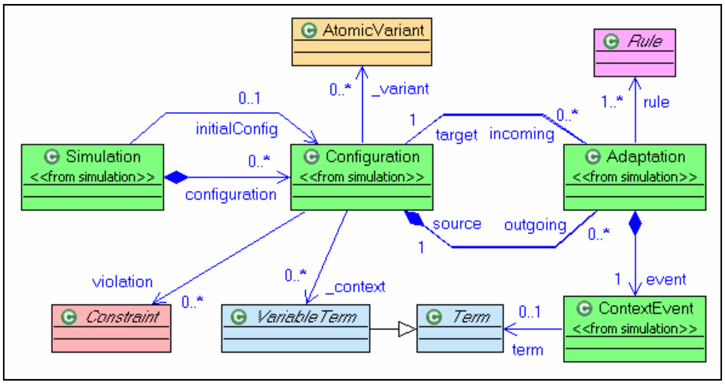


Fig. 7. Simulation model

Based on this simulation model, a prototype simulator has been implemented using the Kermeta platform [12]. The simulator starts from an initial configuration and for each variation of the context variables it evaluates the guards of the adaptation rules. If the guard of an adaptation rule is true in the new context then this rule must be applied and the guards of all the rules are evaluated again. Adaptation rules are applied until none of their guards evaluates to true.

4.2 Simulation Output

The output of a simulation can be rendered as a graph in which each node is a configuration and each edge is an adaptation. Figure 8 shows an excerpt of the simulation graph for the service discovery application. The complete simulation graph for this example contains 24 configurations obtained by aspect weaving and 70 adaptations. In the label of each node, the first line corresponds to the values of the context variables and the second line to the set of aspects that should be used to create the corresponding configuration. Each edge in the graph corresponds to an adaptation to a change of one context variable. The label of the edges starts with the context variable change and details the set of adaptation rules that were applied. In the graph presented in Figure 8 the configurations have been colored in order to visualize easily the battery level. Configurations for which the battery is high are displayed in green and configurations with low battery are displayed in orange.

4.3 Constraint Checking and Rule Termination

The main benefit of the simulation model is to allow for validating the adaptation rules at design-time. As shown in the previous section the adaptation graph can be visualized and colors can be used in order to highlight specific properties. This allows for a manual validation of the specified rules. In addition, the simulation process can identify live-locks and dead-locks in the adaptation graph and allows to automatically verify invariants on the system.

Dead-locks in the simulation graph correspond to cases where some adaptation rules lead to a configuration from which the system cannot adapt. In a design, this could be done voluntarily but in most cases this is due to some incorrect or missing adaptation rules. Live-locks correspond to cases where the system bounces between several configurations while the context is not changing. This situation always reveals

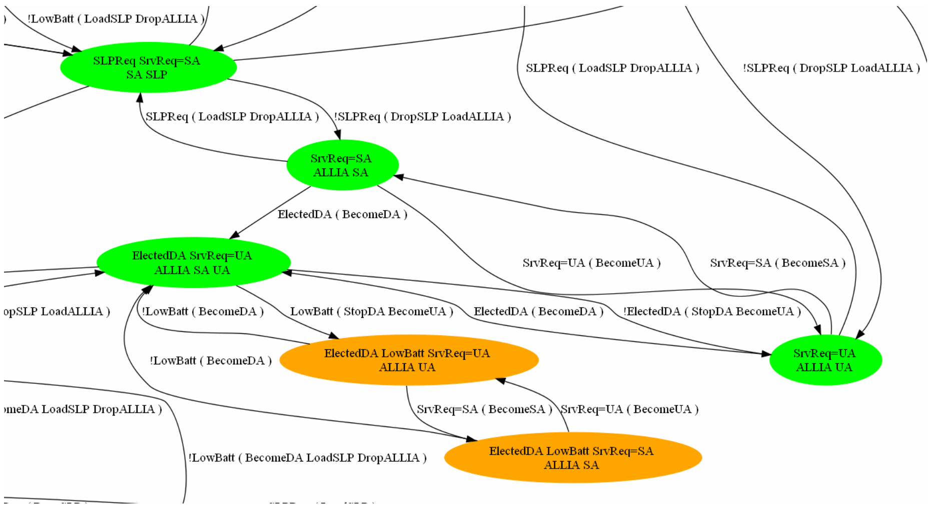


Fig. 8. Excerpt of the simulation graph for the SDA

an error in the adaptation rules. The simulator can identify live-locks while it computes the simulation graph. For a single change of the context, no adaptation rule should be able to apply twice. Indeed, if after applying a rule (and possibly some others), if the same rule can apply again then the rule could be applied an indefinite number of times. When this situation is detected by the simulator, it reports an error in the rules and provides the configuration in which the problem occurs and the sequence of rules which is looping.

The meta-model presented in Section 3 allows defining invariants on the system. These invariants are checked by the simulator on all the configurations that are created during the simulation. Any violation of these invariants reveals an error in the adaptation model.

5 Adapting the System at Runtime

In this section, we present how we actually adapt a running system using the rules we presented in Section 3. In order to trigger the rules, we need to monitor the state of the system itself and the execution context (e.g., memory, CPU usage, available network bandwidth, battery level). For this purpose we intend to reuse the *Intel Mobile Platform Software Development Kit* [1] that already offers a large set of probes. This SDK is freely available and provides a Java API implementing these probes. Using these probes, we have to implement the variables related to the execution context, e.g., *lowBatt*. For example, we can specify that:

```
lowBatt = batteryInstance.percentRemaining < 15
```

However, defining the variable *lowBatt* in this way may be too strict. For example, if the battery level goes under 15%, the system will adapt. But, if the user plugs the system to power supply, the battery level will rapidly increase and the system may adapt again because the battery is not low anymore. In this case, the system adapts twice whereas it would have been preferable to do nothing as the adaptation process may be time consuming.

In order to tackle the instability of rules, we will use WildCAT 2.0, currently still under development. WildCAT [5] is an extensible Java framework that eases the creation of context-aware applications. It provides a simple but yet powerful dynamic model to represent the execution context of a system. The context information can be accessed by two complimentary interfaces: synchronous requests (pull mode: application makes a query on the context) and asynchronous notifications (push mode: context raises information to the application). Internally, it is a framework designed to facilitate the acquisition and the aggregation of contextual data and to create reusable ontologies to represent aspects of the execution context relevant to many applications. A given application can mix different implementations for different aspects of its context while only depending on WildCAT's simple and unified API. The version 2.0 of WildCAT allows defining SQL-like requests on the environment model and integrate the notion of time. For example, it is possible to trigger a rule when the battery has been lower than 15% for more than 3 minutes.

When a rule is triggered, the associated variants become active. In other words, we weave the aspects associated to each variant in the base model. Aspect weaving is currently performed with SmartAdapters [8]. Then, we compare the woven model with the reference model, obtained by introspection over the running system. This comparison generates a diff and match model specifying what has changed in the woven model. By analyzing this model, we automatically generate a safe reconfiguration script that is then applied to the running system. Aspect weaving and automatic adaptation are described in more details in [8, 11].

6 Related Work

Zhang and Cheng present a method for constructing and verifying adaptation models using Petri nets [13]. Different from our approach, Zhang and Cheng identify and define all the configurations in advance. In our approach, we define the context properties, adaptation rules, and variability of the application. The simulator uses this input to eventually produce potential valid system configurations. The simulator identifies system configurations that violate invariants. The simulator also detects live-locks and dead-locks that could be incorrectly triggered by the adaptation rules. Furthermore, the results of the simulation help designers to make decisions when refactoring the system. Also relevant to our work in the DiVA project is the outcome from MADAM/MUSIC [6] projects and SAFRAN related work [4]. The approaches used in MADAM and MUSIC enable runtime adaptation by exploiting architecture models as in our case. SAFRAN is a platform that provides runtime support for adaptive systems. Even though, MADAM, MUSIC, and SAFRAN existing results have inspired our research they do not focus on validation issues.

7 Conclusion and Future Work

This paper presents our ongoing work on modeling adaptation. So far, based on the meta-model we have modeled, simulated and checked a few toy adaptive applications. However, we have also identified the need for more expressiveness in order to describe the variants, the context and the adaptation rules. Our objective is to build on top of the current meta-model in order to identify a restricted set of concepts relevant to the modeling of variability and adaptation. At this stage, we have identified two specific issues.

Firstly, in their current form, the number of adaptation rules can quickly grow as the number of context elements and variants increase. Our main goal is to tackle the problem of an explosive growth in the number of configurations and the artifacts to be used in their construction. However, we do not want to move the complexity associated into the rules as a consequence. Consequently, as a step towards improving our adaptation rules, we aim to express the *rules* using semantics. In that sense, the rule should be of the form "*choose a set of variants with properties that match the current context*". The above embraces a more declarative approach. Although, sometimes we still might want to allow rules on variant configurations since pre-defined full or partial configurations might be extracted or derived from the requirements straightforwardly, as was the case in our variability model.

Secondly, our current simulation prototype enumerates all the configurations and adaptations between them. While this is very useful and works well while the number of configurations is manageable, this approach has the typical model-checking scalability issues when the number of configuration and adaptation grows. Several techniques can be combined in order to keep the simulation space manageable, for example, adding constraints on the context, considering sub-sets of variability dimensions or using heuristics to limit the depth of simulations. In the context of the DiVA project, we plan to experiment with industrial adaptive applications in order to choose the most appropriate solutions to this scalability issue.

For the runtime, as future work we plan to automate as much as possible the implementation of the triggers. For example, it is easy to quantify the domain in which a battery evolves: 0 to 100. But, defining what low level in terms of battery means may be more difficult. We previously said that a battery is low if the remaining percentage is lower than 15 for 3 minutes. However, this kind of information is generally not specified in requirement documents and developers have to infer the information from their knowledge and/or based on experimentation. We plan to use fuzzy logic [9] to help in defining and implementing triggers. Providing a global domain (0 to 100) and some qualifiers ("high", "medium", "low"), the fuzzy logic can determine, for a given observed value (e.g., 17%) if the battery is "low", "medium", etc. Fuzzy logic can help us in filling the gap between requirement (qualitative descriptions) and implementation (quantitative observations) and allows keeping high-level adaptation rules at runtime. Early results are shown in [3].

References

[1] `http://ossmpsdk.intel.com/`

[2] Bencomo, N., France, R., Blair, G.: 2nd international workshop on models@run.time. In: MoDELS 2007 Workshops. LNCS, vol. 5002, pp. 206–211. Springer, Heidelberg (2008)

[3] Chauvel, F., Barais, O., Borne, I., Jezequel, J.-M.: Composition of qualitative adaptation policies. In: Proceedings of Automated Software Engineering Conference (ASE 2008) (2008)

[4] David, P.-C., Ledoux, T.: An aspect-oriented approach for developing self-adaptive fractal components. In: Löwe, W., Südholt, M. (eds.) SC 2006. LNCS, vol. 4089, pp. 82–97. Springer, Heidelberg (2006)

[5] David, P.C., Ledoux, T.: WildCAT: a generic framework for context-aware applications. In: MPAC 2005: 3rd Int. Workshop on Middleware for Pervasive and Ad-hoc Computing, pp. 1–7. ACM, New York (2005)

[6] Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., Gjorven, E.: Using architecture models for runtime adaptability. Software IEEE 23(2), 62–70 (2006)

[7] Flores-Cortés, C.A., Blair, G., Grace, P.: An Adaptive Middleware to Overcome Service Discovery Heterogeneity in Mobile Ad-hoc Environments. IEEE Dist. Systems Online (2007)

[8] Lahire, P., Morin, B., Vanwormhoudt, G., Gaignard, A., Barais, O., Jézéquel, J.-M.: Introducing variability into aspect-oriented modeling approaches. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 498–513. Springer, Heidelberg (2007)

[9] Moon, S., Lee, K.H., Lee, D.: Fuzzy branching temporal logic. IEEE Transactions on Systems Man, and Cybernetics –PartB: Cybernetics 34(2) (2004)

[10] Morin, B., Fleurey, F., Bencomo, N., Jezequel, J.-M., Solberg, A., Dehlen, V., Blair, G.: An aspect-oriented and model-driven approach for managing dynamic variability. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301. Springer, Heidelberg (2008)

[11] Morin, B., Vanwormhoudt, G., Lahire, P., Gaignard, A., Barais, O., Jézéquel, J.-M.: Managing variability complexity in aspect-oriented modeling. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301. Springer, Heidelberg (2008)

[12] Muller, P.A., Fleurey, F., Jézéquel, J.-M.: Weaving executability into object-oriented meta-languages. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 264–278. Springer, Heidelberg (2005)

[13] Zhang, J., Cheng, B.H.C.: Model-based development of dynamically adaptive software. In: International Conference on Software Engineering (ICSE 2006), China (2006)

Model-Based Traces[⋆]

Shahar Maoz

Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Rehovot, Israel
shahar.maoz@weizmann.ac.il

Abstract. We introduce *model-based traces*, which trace behavioral models of a system's design during its execution, allowing to combine model-driven engineering with dynamic analysis. Specifically, we take visual inter-object scenario-based and intra-object state-based models (sequence charts and statecharts) used for a system's design, and follow their activation and progress as they come to life at runtime, during the system's execution. Thus, a system's runtime is recorded and viewed through abstractions provided by behavioral models used for its design. We present two example applications related to the automatic generation and visual exploration of model-based traces and suggest a list of related challenges.

1 Introduction

Transferring model-driven engineering artifacts and methods from the early stages of requirements and specification, during a system's design, to the later stages of the lifecycle, where they would aid in the testing, analysis, maintenance, evolution, comprehension, and manipulation of running programs, is an important challenge in current model-driven engineering research.

In this paper, as a means towards this end, we introduce *model-based traces*, which trace behavioral models from a system's design during its execution, allowing to combine model-driven engineering with dynamic analysis. Specifically, we take visual inter-object scenario-based and intra-object state-based models (sequence diagrams and statecharts) used for a system's design, and follow their activation and progress as they come to life at runtime, during the execution of the system under investigation. Thus, a system's runtime is recorded and viewed through abstractions provided by models used for its design.

An important feature of model-based traces is that they provide enough information to reason about the executions of the system and to reconstruct and replay an execution (symbolically or concretely), exactly at the abstraction level defined by its models. This level of model-based reflection seems to be a necessary requisite for the kind of visibility into a system's runtime required for model-based dynamic analysis and adaptation.

[⋆] This research was supported by The John von Neumann Minerva Center for the Development of Reactive Systems at the Weizmann Institute of Science.

Additional features worth noting. First, model-based traces can be generated and defined based on partial models; the level of abstraction is defined by the modeler. Second, the models used for tracing are not necessarily reflected explicitly in the running program's code; rather, they define a separate viewpoint, which in the process of model-based trace generation is put against the concrete runtime of the program under investigation. Third, the same concrete runtime trace may result in different model-based traces, based on the models used for tracing; and vice versa, different concrete runtime traces may result in identical model-based traces, if the concrete runs are equivalent from the more abstract point of view of the model used for tracing.

In the next section we briefly introduce, informally define, and discuss the format and features of model-based traces, using a simple example. We then present two example applications related to the automatic generation and visual exploration of model-based traces. Finally, we suggest a list of related challenges.

2 Model-Based Traces

The use of system's execution traces for different analysis purposes requires different levels of abstraction, e.g., recording CPU register assignments, recording virtual machine commands, or recording statements at the code level. We suggest a higher level of abstraction over execution traces, based on behavioral models typically used for a system's design, such as sequence diagrams and statecharts.

In this work we present two types of model-based traces, inter-object scenario-based traces and intra-object state-based traces. Additional types may be created by combining variants of the two or using other modeling techniques[1].

Given a program P and a behavioral model M, a model-based execution trace records a run r of P at the level of abstraction induced by M. A unification mechanism is defined, which statically and dynamically maps concrete elements of the run to elements in the model. The type of the model used, the artifacts and their semantics, define the types of entries that appear in the model-based trace. We demonstrate our ideas using two concrete examples of a scenario-based trace and a state-based trace, taken from a small example system.

Note that although there are code generation schemes for the *execution* of the models we use, we do not, in general and in the example given here, consider tracing programs whose code was automatically generated from models. On the contrary, we believe that one of the strengths of our approach is that it can be applied to systems in general, not only to ones where the implementation explicitly reflects certain high-level models.

Also note that the model-based traces we present are not mere projections of the concrete runtime information onto some limited domain. Rather, we use *stateful* abstractions, where trace entries depend on the history and context of the run and the model; the model-based trace not only filters out irrelevant

[1] In principle, any representation of an execution trace may be considered a model-based trace, depending on the definition of what constitutes a model.

information but also adds model specific information (e.g., information about entering and exiting ‘states’ that do not appear explicitly in the program).

A small example. Consider an implementation of the classic PacMan game. PacMan consists of a maze, filled with dots, power-ups, fruit and four ghosts. A human player controls PacMan, who needs to collect as many points as possible by eating the objects in the maze. When a ghost collides with PacMan, it loses a life. When no lives are left, the game is over. However, if PacMan eats a power-up, it is temporarily able to eat the ghosts, thus reversing roles. When a ghost is eaten, it must go back to the jail at the center of the maze before leaving again to chase PacMan. When all dots are eaten, the game advances to the next – more difficult – level. We consider the PacMan game to be a well-known, intuitive, relatively small and yet complex enough reactive system, hence a good choice for the purpose of demonstrating model-based traces in this paper.

2.1 Scenario-Based Models

For inter-object scenario-based modeling, we use a UML2-compliant variant of Damm and Harel’s *live sequence charts* (LSC) [4,9]. Roughly, LSC extends the partial order semantics of sequence diagrams in general with a universal interpretation and must/may (hot/cold) modalities, and thus allows to specify scenario-based liveness and safety properties. Must (hot) events and conditions are colored in red and use solid lines; may (cold) events and conditions are colored in blue and use dashed lines. A specification typically consists of many charts, possibly interdependent, divided between several use cases (our small PacMan example has 9 scenarios divided between 3 use cases).

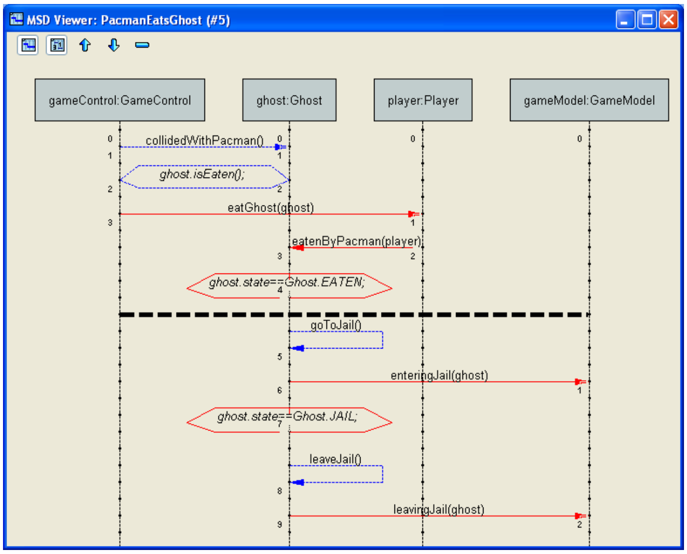


Fig. 1. The LSC for `PacManEatsGhost` with a cut displayed at (3,4,2,0)

Fig. 1 shows one LSC taken from our example model of PacMan. Vertical lines represent specific system objects and time goes from top to bottom. Roughly, this scenario specifies that *"whenever a `gameControl` calls a ghost's `collidedWithPacman()` method and the ghost's `isEaten()` method evaluates to TRUE, the `gameControl` must tell the `player` (PacMan) to eat the `ghost`, the `player` must tell the `ghost` it has been eaten, and the ghost's `state` must change to `EATEN`. Then, if and when the `ghost` goes to jail it must tell the `gameModel` it has gone there and its `state` should change to `JAIL`, etc..."* Note the use of hot 'must' elements and cold 'may' elements. Also, note the use of symbolic instances (see [15]): the lifeline representing `ghost` may bind at runtime to any of the four ghosts (all four are instances of the class `Ghost`).

An important concept in LSC semantics is the *cut*, which is a mapping from each lifeline to one of its locations (note the tiny location numbers along the lifelines in Fig. 1, representing the state of an active scenario during execution). The cut `(3,4,2,0)`, for example, comes immediately after the hot evaluation of the ghost's state. A cut induces a set of enabled events — those immediately after it in the partial order defined by the diagram. A cut is hot if any of its enabled events is hot (and is cold otherwise). When a chart's minimal event occurs, a new instance of it is activated. An occurrence of an enabled method or `true` evaluation of an enabled condition causes the cut to progress; an occurrence of a non-enabled method from the chart or a `false` evaluation of an enabled condition when the cut is *cold* is a *completion* and causes the chart's instance to close gracefully; an occurrence of a non-enabled method from the chart or a `false` evaluation of an enabled condition when the cut is *hot* is a *violation* and should never happen if the implementation is faithful to the specification model. A chart does not restrict events not explicitly mentioned in it to occur or not to occur during a run (including in between events mentioned in the chart).

2.2 Scenario-Based Traces

Given a scenario-based specification consisting of a number of LSCs, a *scenario-based trace* includes the activation and progress information of the scenarios, relative to a given program run. A trace may be viewed as a projection of the full execution data onto the set of methods in the specification, plus, significantly, the activation, binding, and cut-state progress information of all the instances of the charts (including concurrently active multiple copies of the same chart). Thus, our scenario-based traces may include the following types of entries:

– **Event occurrence.** representing the occurrence of an event. Events are timestamped and are numbered in order of occurrence. Only the events that explicitly appear in one of the scenarios in the model are recorded in the trace (one may add identifiers of participating objects, i.e., caller and callee, and parameter values). The format for an event occurrence entry is:

  ```
  E: <timestamp> <event no.>: <event signature>
  ```

– **Binding.** representing the binding of a lifeline in one of the active scenario instances to an object. Its format is:

```
...
E: 1172664920526 64: void pacman.classes.Ghost.slowDown()
B: PowerUpEaten[1] lifeline 6 <- pacman.classes.Ghost@7e987e98
B: GhostStopsFleeing[7] lifeline 1 <- pacman.classes.Ghost@7e987e98
C: GhostStopsFleeing[7] (0,1) Hot
C: GhostFleeing[7] (1,3) Hot
E: 1172664920526 65: void pacman.classes.GameControl.ghostSlowedDown(Ghost) pacman.classes.Ghost@7e987e98
B: GhostStopsFleeing[7] lifeline 0 <- pacman.classes.GameControl[panel0,0,0,600x600,layout=...
C: GhostStopsFleeing[7] (1,2) Cold
C: GhostFleeing[7] (2,4) Cold
E: 1172664920526 66: void pacman.classes.GameModel.resetGhostPoints()
C: PowerUpEaten[1] (1,2,6,1,1,1,1) Cold
F: PowerUpEaten[1] Completion
E: 1172664921387 67: void pacman.classes.Fruit.enterScreen()
B: PacmanEatsFruit[0] lifeline 2 <- pacman.classes.Fruit@3360336
C: PacmanEatsFruit[0] (0,0,1,0) Hot
C: PacmanEatsFruit[0] (0,0,2,0) Cold
E: 1172664923360 68: void pacman.classes.Ghost.collidedWithPacman()
B: PacmanEatsGhost[2] lifeline 1 <- pacman.classes.Ghost@7d947d94
B: PacmanEatsGhost[2] lifeline 0 <- pacman.classes.GameControl[panel0,0,0,600x600,layout=...
C: PacmanEatsGhost[2] (1,1,0,0) Hot
C: PacmanEatsGhost[2] (1,2,0,0) Hot
C: GhostEatsPacman[2] (0,1,1,0) Cold
F: GhostEatsPacman[2] Violation
...
```

Fig. 2. Part of a textual representation of a scenario-based trace of PacMan

```
B: <scenario name>[instance no.] lifeline <no.> <- <object identifier>
```

– **Cut change.** representing a cut change in one of the active scenario instances. Its format is:

  ```
  C: <scenario name>[instance no.] <cut tuple> [Hot|Cold]
  ```

– **Finalization.** representing a successful completion or a violation in an active scenario instance. Its format is:

  ```
  F: <scenario name>[instance no.] [Completion|Violation]
  ```

Fig. 2 shows an example short snippet from a scenario-based trace of PacMan. Note the different types of entries that appear in the trace.

2.3 State-Based Models

For intra-object state-based modeling, we use UML state machines (that is, the object based variant of Harel statecharts [7]). For lack of space, we assume the reader is partly familiar with the syntax and semantics of statecharts in general, at least to the level that allows to understand our example.

Fig. 3 shows an example statechart taken from a model of PacMan. It shows part of a statechart for the class `Ghost`.

2.4 State-Based Traces

Given a state-based specification consisting of a number statecharts, a *state-based trace* includes the creation and progress information of the statecharts, relative to a given program run. The trace includes information on events, guards evaluation, and the entering and exiting of states in all instances of the statecharts (including concurrently running instances of the same statechart). Thus, our state-based traces may include the following types of entries:

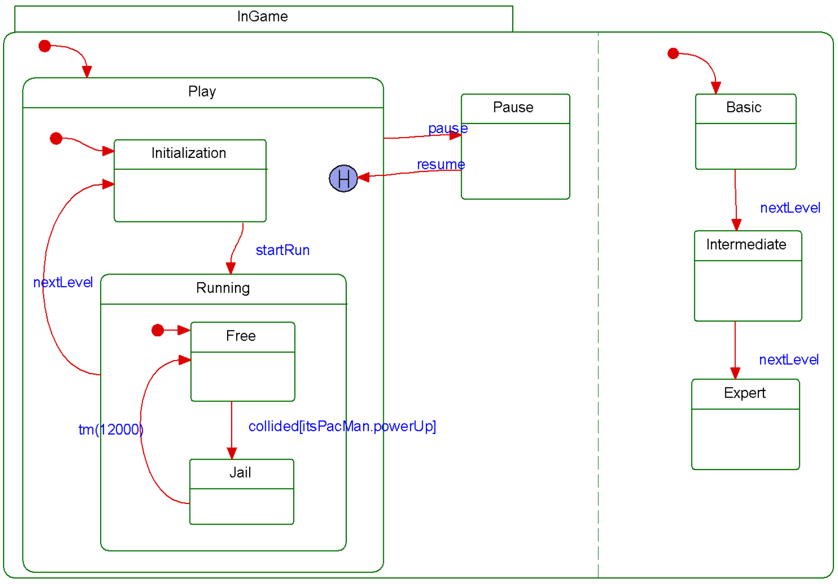


Fig. 3. Part of the `Ghost` statechart in the PacMan model

```
...
EV: 45632290 874: Ghost[3].collided
EX: Ghost[3] Exited state Ghost.InGame.InPlay.Play.Running.Free
EN: Ghost[3] Entered state Ghost.InGame.InPlay.Play.Running.Jail
EV: 45644272 875: Ghost[2].collided
EX: Ghost[2] Exited state Ghost.InGame.InPlay.Play.Running.Free
EN: Ghost[2] Entered state Ghost.InGame.InPlay.Play.Running.Jail
EV: 45644290 876: Ghost[3].timer
EX: Ghost[3] Exited state Ghost.InGame.InPlay.Play.Running.Jail
EN: Ghost[3] Entered state Ghost.InGame.InPlay.Play.Running.Free
EV: PacMan[1] 877: Pacman[1].complete
EX: PacMan[1] Exited state PacMan.InPlay.Play
EN: PacMan[1] Entered state PacMan.InPlay.LevelInitalization
EV: 45664403 878: Ghost[1].nextLevel
EX: Ghost[1] Exited state Ghost.InGame.InPlay.Play.Running.Free
EX: Ghost[1] Exited state Ghost.InGame.Levels.Basic
EN: Ghost[1] Entered state Ghost.InGame.InPlay.Play.Initalization
EN: Ghost[1] Entered state Ghost.InGame.Levels.Intermediate
EV: 45664405 879: Ghost[2].nextLevel
EX: Ghost[2] Exited state Ghost.InGame.InPlay.Play.Running.Jail
EX: Ghost[2] Exited state Ghost.InGame.Levels.Basic
EN: Ghost[2] Entered state Ghost.InGame.InPlay.Play.Initalization
EN: Ghost[2] Entered state Ghost.InGame.Levels.Intermediate
EV: 45664408 880: Ghost[3].nextLevel
...
```

Fig. 4. Part of a textual representation of a state-based trace of PacMan

– **State entered** representing a statechart entering a state. The format is:

  `EN: <class_name>[instance no.] Entered state <state full name>`

– **State exited** representing a statechart existing a state. The format is:

  `EX: <class_name>[instance no.] Exited state <state full name>`

– **Event occurrence** representing the occurrence of an event. Events are timestamped and are numbered in order of occurrence. Only the events that explicitly appear in one of the statecharts in the model are recorded in the trace. One may optionally add guards evaluation. The format is:

```
EV: <timestamp> <event no.>: <event signature>
```

Fig. 4 shows a snippet from a state-based trace of PacMan involving a number of statecharts. Note the different types of entries that appear in the trace.

We remark that the above scenario-based and state-based trace formats are presented as examples. Depending on the application, the trace generation mechanism available, and the kind of analysis and reasoning intended for the model-based traces, one may consider different formats, different entry types, different levels of succinctness etc. For example, whether to document the values of guards or the concrete values of parameters depends on the specific application and expected usage of the model-based trace.

3 Example Applications

We give a short overview of two example applications related to the generation of model-based traces and to their visualization and exploration.

3.1 Generating Model-Based Traces

S2A [8] (for Scenarios to Aspects) is a compiler that translates live sequence charts, given in their UML2-compliant variant using the *modal* profile [9], into AspectJ code [1], and thus provides full code generation of reactive behavior from visual declarative scenario-based specifications. S2A implements a compilation scheme presented in [13]. Roughly, each sequence diagram is translated into a *scenario aspect*, implemented in AspectJ, which simulates an automaton whose states correspond to the scenario cuts; transitions are triggered by AspectJ pointcuts, and corresponding advice is responsible for advancing the automaton to the next cut state.

Most important in the context of this paper, though, is that in addition to scenario-based execution (following the play-out algorithm of [10]), S2A provides a mechanism for scenario-based monitoring and runtime verification. Indeed, the example scenario-based trace shown in Fig. 2 is taken from an actual execution log of a real Java program of the PacMan game adapted from [3], (reverse) modeled using a set of live sequence charts (drawn inside IBM Rational SA [2] as modal sequence diagrams), and automatically instrumented by the AspectJ code generated by S2A. More on S2A and its use for model-based trace generation can be found in `http://www.wisdom.weizmann.ac.il/~maozs/s2a/`.

3.2 Exploring Model-Based Traces

The Tracer [14] is a prototype tool for the visualization and interactive exploration of model-based traces. The input for the Tracer is a scenario-based model of a

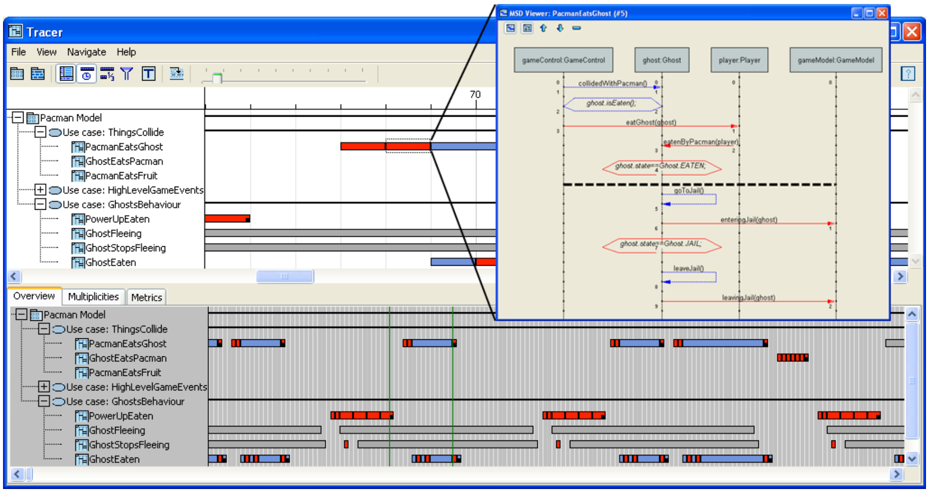


Fig. 5. The Tracer's main view, an opened scenario instance with its cut displayed at (3,4,2,0), and the Overview pane (at the bottom). The example trace and model are taken from an implementation of the PacMan game, see [14].

system given as a set of UML2-compliant live sequence charts, and a scenario-based trace, generated from an execution of the system under investigation.

Fig. 5 shows a screenshot of the main view of the Tracer, displaying a scenario-based model and trace similar to the one shown in Fig. 2. Roughly, the main view is based on an extended hierarchical Gantt chart, where time goes from left to right and a two-level hierarchy is defined by the containment relation of use cases and sequence diagrams in the model. Each leaf in the hierarchy represents a sequence diagram, the horizontal rows represent specific active instances of a diagram, and the blue and red bars show the duration of being in a specific cold and hot relevant cuts. The horizontal axis of the view allows to follow the progress of specific scenario instances over time, identify events that caused progress, and locate completions and violations. The vertical axis allows a clear view of the synchronic characteristic of the trace, showing exactly what goes on, at the models abstraction level, at any given point in time.

When double-clicking a bar, a window opens, displaying the corresponding scenario instance with its dynamic cut shown in a dashed black line. Identifiers of bound objects and values of parameters and conditions are displayed in tooltips over the relevant elements in the diagram. In addition, one can travel back and forth along the cuts of the specific instance (using the keyboard or the arrows in the upper-left part of the window). Multiple windows displaying the dynamic view of different scenario instances can be opened simultaneously to allow for a more global synchronic (vertical) view of a specific point in the execution, or for a diachronic (horizontal) comparison between the executions of different instances of the same scenario at different points in time during the execution.

Note the Overview pane (bottom of Fig. 5), which displays the main execution trace in a smaller pixel per event scale, and the moving window frame showing the borders of the interval currently visible in the main view. The Overview allows to identify high level recurring behavioral patterns, at the abstract level of the scenarios in the model. Additional views are available, supporting multiplicities, event-based and real-time based tracing, and the presentation of various synchronous metrics (e.g., how many scenarios have been affected by the most recent event?). Overall, the technique links the static and dynamic aspects of the system, and supports synchronic and diachronic trace exploration. It uses overviews, filters, details-on-demand mechanisms, multi-scaling grids, and gradient coloring methods.

The Tracer was first presented in [14]. More on the Tracer, including additional screenshots and screencasts can be found in `http://www.wisdom.weizmann.ac.il/~maozs/tracer/`.

4 Related Work

We briefly discuss related work. Generating model-based traces requires an observer with monitoring and decision-making capabilities; a so called ‘runtime awareness’ component (see, e.g., [5,11]). However, while model-based traces can be used for error detection and runtime verification, the rich information embedded in them supports more general program comprehension and analysis tasks and allows the reconstruction and symbolic replay of a program’s run at the abstraction level defined by the model used for tracing.

The use of AOP in general and AspectJ in particular to monitor program behavior based on behavioral properties specified in (variants of) LTL has been suggested before (see, e.g., [5,16]). As LSCs can be translated into LTL (see [12]), these work have similarities with our use here of S2A. Like [16], S2A automatically generates the AspectJ code which simulates the scenario automaton (see [13]). Unlike both work however, S2A outputs a rich trace reflecting state changes and related data (binding etc.), to serve our goal of generating model-based traces that allow visibility and replaying, not only error detection.

Many work suggest various trace generation and visual exploration techniques (e.g., for a survey, see, [6]). Most consider code level concrete traces. Some attempt to extract models from these traces. In contrast, model-based traces use an abstraction given by user-defined models. They are generated by symbolically running these models simultaneously with a concrete program execution.

5 Discussion and Challenges for Future Work

We introduced model-based traces and presented two example applications. The focus of model-based traces is on providing visibility into an execution of a program at the abstraction level defined by a model, enabling a combination of dynamic analysis and model-driven engineering. Below we discuss our approach and list related challenges.

Trace generation. S2A provides an example of a model-based trace generation technology, based on programmatically generated aspects. Two major advantages of this approach are that the monitoring code is automatically generated from the models, and that the code of the system under investigation itself is oblivious to the models 'watching' it. Related challenges include minimizing runtime overhead, scalability in terms of trace length and model size, and the application of similar technology to domains where aspect technology is not readily available (e.g., various embedded or distributed systems).

Analysis and reasoning. We consider the development of analysis methods for model-based traces. For example, define and measure various vertical and horizontal metrics (e.g., 'bandwidth', state / transition coverage per trace per model, how many times was each state visited), abstraction and refinement operators (e.g., hide events and keep states, hide sub states of composite states), ways to represent and compare different model-based runtime configurations ('snapshots', perhaps most important for dynamic adaptation), or ways to align and compare between traces of different runs of the same system, or very similar runs of different versions of the same system. In addition, we consider additional types of abstractions over the same models, e.g., real-time based vs. event-based trace representation (as is supported by the Tracer (see [14])). Also, an agreeable, common representation format for model-based traces, where applicable (e.g., for specific types of models), should perhaps be defined and agreed upon, so that not only models but also their traces may be exchanged between tools in a standard format like XMI.

Visualization and interaction. The visualization and interaction supported by the Tracer allows a human user to explore and analyze long and complex model-based traces that are otherwise very hard to handle manually in their textual form. Still, a lot more can be done on this front, from finding "economic" visualizations for model-based snapshots to animation to visual filters etc.

Acknowledgements. I would like to thank David Harel, David Lo, Itai Segall, Yaki Setty, and the anonymous reviewers for comments on a draft of this paper.

References

1. AspectJ, `http://www.eclipse.org/aspectj/`
2. IBM Rational Software Architect, `http://www-306.ibm.com/software/awdtools/architect/swarchitect/`
3. PacMan. Java implementation of the classic PacMan game, `http://www.bennychow.com`
4. Damm, W., Harel, D.: LSCs: Breathing Life into Message Sequence Charts. J. on Formal Methods in System Design 19(1), 45–80 (2001)
5. Goldsby, H.J., Cheng, B.H.C., Zhang, J.: AMOEBA-RT: Run-Time Verification of Adaptive Software. In: Giese, H. (ed.) MoDELS 2007 Workshops. LNCS, vol. 5002, pp. 212–224. Springer, Heidelberg (2008)
6. Hamou-Lhadj, A., Lethbridge, T.C.: A Survey of Trace Exploration Tools and Techniques. In: Lutfiyya, H., Singer, J., Stewart, D.A. (eds.) Proc. Conf. of the Centre for Advanced Studies on Collaborative research (CASCON 2004), pp. 42–55. IBM (2004)

7. Harel, D., Gery, E.: Executable Object Modeling with Statecharts. IEEE Computer 30(7), 31–42 (1997)
8. Harel, D., Kleinbort, A., Maoz, S.: S2A: A Compiler for Multi-modal UML Sequence Diagrams. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 121–124. Springer, Heidelberg (2007)
9. Harel, D., Maoz, S.: Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. Software and Systems Modeling (SoSyM) 7(2), 237–252 (2008)
10. Harel, D., Marelly, R.: Specifying and executing behavioral requirements: the play-in/play-out approach. Software and Systems Modeling (SoSyM) 2(2), 82–107 (2003)
11. Hooman, J., Hendriks, T.: Model-Based Run-Time Error Detection. In: Giese, H. (ed.) MODELS 2008. LNCS, vol. 5002, pp. 225–236. Springer, Heidelberg (2008)
12. Kugler, H.J., Harel, D., Pnueli, A., Lu, Y., Bontemps, Y.: Temporal Logic for Scenario-Based Specifications. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 445–460. Springer, Heidelberg (2005)
13. Maoz, S., Harel, D.: From Multi-Modal Scenarios to Code: Compiling LSCs into AspectJ. In: Young, M., Devanbu, P.T. (eds.) FSE 2006, pp. 219–230. ACM, New York (2006)
14. Maoz, S., Kleinbort, A., Harel, D.: Towards Trace Visualization and Exploration for Reactive Systems. In: Proc. IEEE Symp. on Visual Languages and Human-Centric Computing (VL/HCC 2007), pp. 153–156. IEEE Computer Society, Los Alamitos (2007)
15. Marelly, R., Harel, D., Kugler, H.: Multiple Instances and Symbolic Variables in Executable Sequence Charts. In: Proc. 17th ACM Conf. on Object-Oriented Prog., Systems, Lang. and App (OOPSLA 2002), pp. 83–100 (2002)
16. Stolz, V., Bodden, E.: Temporal Assertions using AspectJ. Electr. Notes Theor. Comput. Sci. 144(4), 109–124 (2006)

Model Co-evolution and Consistency Management (MCCM'08)

Dirk Deridder[1], Jeff Gray[2], Alfonso Pierantonio[3], and Pierre-Yves Schobbens[4]

[1] Vrije Universiteit Brussel, Belgium
dirk.deridder@vub.ac.be
[2] University of Alabama at Birmingham, USA
gray@cis.uab.edu
[3] Università degli Studi dell'Aquila, Italy
alfonso@di.univaq.it
[4] Université de Namur, Belgium
pierre-yves.schobbens@fundp.ac.be

Abstract. The goal of the workshop was to exchange ideas and experiences related to Model (Co-)evolution and Consistency Management (MCCM) in the context of Model-Driven Engineering (MDE). Contemporary MDE practices typically include the manipulation and transformation of a large and heterogeneous set of models. This heterogeneity exhibits itself in different guises ranging from notational differences to semantic content-wise variations. These differences need to be carefully managed in order to arrive at a consistent specification that is adaptable to change. This requires a dedicated activity in the development process and a rigourous adoption of techniques such as model differencing, model comparison, model refactoring, model (in)consistency management, model versioning, and model merging. The workshop invited submissions from both academia and industry on these topics, as well as experience reports on the effective management of models, metamodels, and model transformations. We selected ten high-quality contributions out of which we included two as best-papers in the workshop reader. As a result of the high number of participants and the nice mix of backgrounds we were able to debate lively over a number of pertinent questions that challenge our field.

1 Introduction

In general, software artifacts and applications are subject to many kinds of changes. These range from technical changes due to rapidly evolving technology platforms, to modifications resulting from the natural evolution of the business that is supported. This includes changes at all levels, from requirements through architecture and design, to source code, documentation and test suites. They typically affect various kinds of models including data models, behavioral models, domain models, source code models, goal models, etc. Coping with and managing the changes that accompany the evolution of software assets is therefore an essential aspect of Software Engineering as a discipline.

Model-Driven Engineering (MDE) is an approach to software design and development in which models are the primary artifacts of software development.

The major objective of MDE is to increase productivity and reduce time-to-market by raising the level of abstraction. In part this is done by using concepts closer to the problem domain instead of those offered by programming languages. Such models represent domain-specific concepts and conform to metamodels.

A core task of MDE is the manipulation and transformation of models. Thus, Model (Co-)evolution and Consistency Management (MCCM) are crucial activities to cope with the natural changes of the corresponding software system. Currently, there is an increasing need for more disciplined techniques and engineering tools to support a wide range of model evolution activities, including model differencing, model comparison, model refactoring, model inconsistency management, model versioning and model merging.

Recently, a number of works devoted to the detection of differences between models has emerged to foster enhanced model management practices. The exploitation of differences is an appropriate solution for version management, because in general the complete system model is far larger than the modifications that occur from one version to another. Apart from these works, further research is required to address the rest of the model evolution activities (refactoring, inconsistency management, versioning, etc.). Moreover, the different dimensions of evolution make the problem intrinsically difficult because modifications can reflect coherent adaptations of correlated artifacts at several layers of the metamodeling architecture. For example, some well-formedness rules can be invalidated when a metamodel evolves. The same happens with the associated model transformations. Furthermore, model adaptations should be propagated to artifacts interconnected by means of model transformations. Finally, the evolution of model transformations should be reflected in both source and target models.

In addition, there is a substantial difference between the modeling of evolution and the evolution of models. There are plenty of works on the former topic, while our proposed workshop focuses on the evolution of models. One of the goals of this workshop was to explain and clarify the difference between these two notions, by explicitly identifying the concepts and mechanisms involved in each one. In particular, we targeted the cross-fertilization of both the MDE and software evolution communities. This is why we considered models in a very broad sense to allow researchers from different communities to identify and discuss commonalities/differences among their specific MCCM problems.

2 About the Workshop

The workshop was a great success[1]. We had 59 registered participants which resulted in many interesting and lively discussions. In total we accepted 10 submissions for presentation which are listed below (presenters are underlined). The contributions marked with an **(*)** were selected as best-papers and are included in the Workshop Reader. The other papers are available in the electronic workshop proceedings[2].

[1] MCCM'08 was organised in cooperation with the MoVES network, funded by the Belgian State, Belgian Science Policy `http://moves.vub.ac.be/`

[2] MCCM'08 Workshop Proceedings: `http://www.info.fundp.ac.be/mccm/`

An Inconsistency Handling Process,
by Ragnhild Van Der Straeten

Retainment Rules for Model Transformations,
by Thomas Goldschmidt and Axel Uhl

(*) Triple Graph Grammars or Triple Graph Transformation Systems? A Case Study from Software Configuration Management,
by Bernhard Westfechtel, Thomas Buchmann and Alexander Dotor

MOD2-SCM: Experiences with Co-evolving Models when Designing a Modular SCM System,
by Thomas Buchmann, Alexander Dotor and Bernhard Westfechtel

Efficient Recognition and Detection of Finite Satisfiability Problems in UML Class Diagrams: Handling Constrained Generalization Sets, Qualifiers and Association Class Constraints,
by Azzam Maraee, Victor Makarenkov Makarenkov and Mira Balaban

COPE: A Language for the Coupled Evolution of Metamodels and Models,
by Markus Herrmannsdoerfer, Sebastian Benz and Elmar Juergens

(*) On Integrating OCL and Triple Graph Grammars,
by Duc-Hanh Dang and Martin Gogolla

Model Engineering using Multimodeling,
by Christopher Brooks, Chih-Hong Cheng, Thomas Huining Feng, Edward A. Lee and Reinhard von Hanxleden

AMOR Towards Adaptable Model Versioning,
by Kerstin Altmanninger, Gerti Kappel, Angelika Kusel, Werner Retschitzegger, Martina Seidl, Wieland Schwinger and Manuel Wimmer

Co-Evolution and Consistency in Workflow-based Applications,
by Mario Sanchez, Jorge Villalobos and Dirk Deridder

After the presentations we scheduled a number of plenary discussions. Each discussion was focused on a particular question that addressed a major theme in model consistency management and model (co-)evolution.

What are the steps required to install a full-fledged MCCM process?
The need for having a rigourous consistency and co-evolution process in place was widely acknowledged. A number of participants pointed out that setting up and (technologically) supporting an MDE process is already a big challenge. It was agreed upon that more effort should go into defining an integral approach.

What are the MCCM challenges when managing large models?
Several issues were discussed related to working with large models (both in terms of size and diversity). In particular, the scalability of existing consistency handlers was questioned, this not only with respect to the computational side but also to the possible cascade of inconsistencies. A challenge in the future will be to derive which inconsistencies to address first in order to reduce the size of problems reported. In addition, it is desirable to guide developers to cope with the 'unstructured' set of inconsistencies (e.g., by

tagging the inconsistencies with their type and importance). Also, the need to temporarily tolerate inconsistencies was identified, which requires a mechanism to reason with inconsistent models (e.g., by using a kind of 'closest semantic match' to overcome problems).

How does bridging semantic domains impact MCCM?
The main issue discussed was how inconsistency handling techniques are challenged by having to address model elements that stem from different semantic universes. It was suggested that we also need dedicated formalisms to specify cross-domain incompatibilities. This is in line with the growing tendency towards domain-specific modeling languages, and it requires to balance the benefits of domain-independent support versus dedicated support (e.g., dedicated detection engines, formalisms, resolution strategies).

What are the different perspectives on MCCM and how can we learn from each other?
One of the goals of the workshop was to assemble researchers from diverse fields in which MCCM is being addressed. We believe this goal was met as exemplified by the contributions of the participants. Some participants discussed the topic from a generic MDE point of view, whereas others discussed it specific to a particular application domain (e.g., software configuration management, workflow-based applications). Also, in terms of the proposed techniques we had a good distribution of topics (e.g., retainment rules, triple graph grammars, ocl, coupled evolution of metamodels and models). As a result, we were able to hold a lively debate during which we focused on identifying and relating the commonalities and differences of the different perspectives. There was a general consensus that the community would benefit from bringing together existing categorisations of co-evolution scenarios and (in)consistency types. Additionally, it was suggested to record the differences in terminology since there are often subtle semantic deltas that might cause misunderstandings (e.g., co-evolution versus coupled evolution versus co-adaptation). During the workshop we were already able to clarify a number of misconceptions. The alignment of terminology was also useful to identify how we might possibly benefit from each other's techniques and approaches. Therefore, a number of workshop participants suggested to set up an online community in the form of a wiki. It is our hope that this wiki will further enable the cross-fertilisation of both the MDE and software evolution communities with respect to consistency and (co-)evolution problems.

Acknowledgements

We thank our programme committee members for their help in the paper selection process: Jean Bézivin, Rubby Casallas, Antonio Cicchetti, Serge Demeyer, Stéphane Ducasse, Vincent Englebert, Jean-Marie Favre, Tudor Gîrba, Reiko Heckel, Viviane Jonckers, Frederic Jouault, Ralf Lämmel, Kim Mens, Tom Mens, Jonathan Sprinkle, Antonio Vallecillo, Ragnhild Van Der Straeten. We also thank Anthony Cleve and Andreas Classen for the considerable effort they put into the practical organisation of MCCM'08.

On Integrating OCL and Triple Graph Grammars

Duc-Hanh Dang and Martin Gogolla

Department of Computer Science, University of Bremen
D-28334 Bremen, Germany
{hanhdd,gogolla}@informatik.uni-bremen.de

Abstract. Triple Graph Grammars (TGGs) tend to be a promising approach for explaining relationships between models in general, and model co-evolution and model consistency within model-driven development in particular. Declarative TGG descriptions can be translated into operational scenarios for model integration, model synchronization, and model transformation. To realize such scenarios, restrictions formulated with the Object Constraint Language (OCL) are an important factor. How to integrate TGGs and OCL is a topic of ongoing research activities. There are strong similarities between the result of such an integration and the Queries, Views and Transformations (QVT) standard of the Object Management Group (OMG). We propose a language for this integration: One part of this language has a one-one mapping to TGGs and the remaining part covers OCL concepts. The language is realized in our tool UML-based Specification Environment (USE) by taking two views on operations derived from TGG rules: Declarative OCL pre- and postconditions are employed as operation contracts, and imperative command sequences are taken as an operational realization. USE offers full OCL support for restricting models and metamodels with invariants, for checking pre- and postconditions of operations as well as for validating and animating transformation operations. Our approach covers a complete realization of TGGs incorporating OCL within our tool USE.

1 Introduction

Triple graph grammars (TGGs) have been first formally proposed in [1]. Their aim was to ease the description of complex transformations within software engineering. The original TGGs extend earlier ideas from [2]. With the advent of model-centric software development approaches (like UML) and model-driven engineering (MDE), TGGs tend to become one formal basis for underlying development concepts. TGGs and the recent QVT (Queries, Views, Transformations) share many functions and building blocks. But in contrast to TGGs, QVT includes the declarative language OCL (Object Constraint Language) which allows to express properties and to navigate in complex models.

In the recent literature, a need for an integration of TGGs and OCL has been expressed. However, such an integration has not been systematically studied or realized yet. Recent papers [3,4,5,6,7] often either underestimate the importance of OCL for constraints as pre- and postconditions of TGG rules or concentrate on non-deleting rules or ignore the automatic construction of metamodels for the correspondence domain of TGG rules. Within these works, OCL constraints are only employed for updating attribute expressions, whereas we also support OCL pre- and postconditions in TGG rules and OCL (meta-)-model invariants.

We propose a language for the integration of TGGs and OCL: One part of this language has a one-one mapping to TGGs and the remaining part covers OCL concepts. The language is realized in our tool UML-based Specification Environment (USE) [8] by taking two views on operational scenarios derived from a TGG rule: OCL pre- and postconditions are employed as operation contracts, and command sequences are taken as an operational realization. Our approach covers a complete realization of TGGs incorporating OCL within our tool USE as well as an on-the-fly verification of the operations.

The rest of this paper is structured as follows. Section 2 explains the basic idea by means of an example. Section 3 describes our realization in USE for the integration. Section 4 explains how the integration of TGGs and OCL can support model co-evolution and consistency. Section 5 comments on related work. The paper is closed with a summary and a conclusion.

2 The Basic Idea for Integrating TGGs and OCL

This section explains the basic idea of our approach by means of an example. We consider a typical situation within software development, where one wants to formally trace the relationship between a conceptual model and a design model. Figure 1 indicates the correspondence between an association end on the left side within a conceptual model and an attribute and an operation on the right side within a design model. For example, we recognize that the association end *customer* is mapped to the attribute *customer* and the operation *getCustomer*() of the class *Order*.

We want to express the requirement that, when we add an association between *Customer* and *Order* in the conceptual model, the corresponding classes

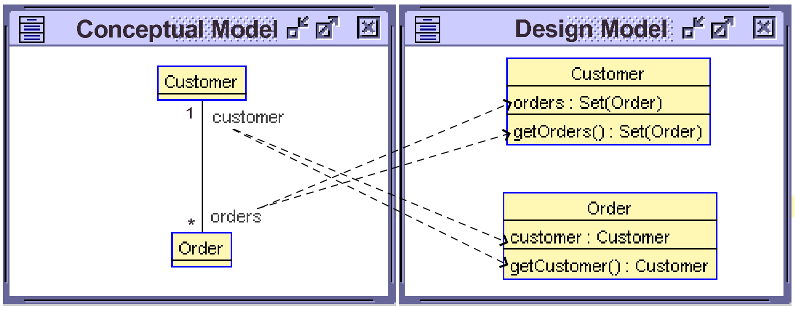


Fig. 1. Correspondence between Association Ends and Attributes

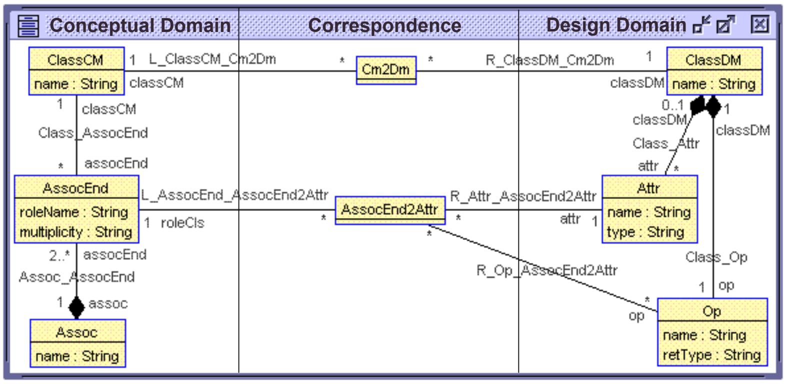


Fig. 2. Simplified Example Metamodels and their Connection

in the design model are automatically extended with new attributes and operations. Therefore, a correspondence between the association ends in the conceptual model and the attributes and operations in the design model has to be established in a formal way.

This scenario is a model transformation between UML class diagrams, and QVT [9] as well as TGGs can approach the requirement. Figure 2 shows the transformation in schematic form as a metamodel. The left side shows the metamodel for the conceptual domain, the right side displays the metamodel for the design domain, and the middle part pictures the connection resp. the correspondence between them. Later we will see, that the diagram in Fig. 1 can be seen as an instance diagram for the metamodels and that the dashed arrows (from Fig. 1) are represented as objects belonging to classes from the middle part. A TGG rule as well as QVT mappings can realize an action which adds an association and updates the attributes and operations as explained in the following.

2.1 QVT for the Example Transformation

Figure 3 presents the example transformation in the QVT Core language. The transformation contains two mappings, *cm2Dm* and *assocEnd2Attr*. The mappings declare constraints that must hold between the model elements within the transformation. The transformation requires two new types of elements (two new classes) for the correspondence domain: *Cm2Dm* and *AssocEnd2Attr*. The mappings can be executed by matching patterns which are declared within the '`check`' parts, i.e., domain patterns, and the '`where`' part, i.e., middle patterns. Each pattern includes a guard part and a bottom part.

The mapping *cm2Dm* allows to access a set of object pairs of the conceptual model and the design model, which coincide in their name.

In the mapping *assocEnd2Attr*, the first '`check`' part means that when two *ClassCM* objects in the guard part are defined, the restriction including OCL constraints in the bottom part must hold. The second '`check`' part means that when two objects of *ClassDM* in the guard part are defined, two *Attr* objects

```
class Cm2Dm{
 classCM: ClassCM;
 classDM: ClassDM;
}


class AssocEnd2Attr{
 role: AssocEnd;
 op:Op;
 attr: Attr;
}
--------------------

map cm2Dm{
 check cm(){
  classCM: ClassCM;
 }
 check dm(){
  classDM: ClassDM;
 }
 where(){cm2dm:Cm2Dm |
  cm2dm.classCM=classCM;
  cm2dm.classDM=classDM;
  classCM.name
          =classDM.name;
 }}
```

```
map assocEnd2Attr{
 check cm(oneCM:ClassCM; manyCM:ClassCM){
  oneRole:AssocEnd,assoc:Assoc,manyRole:AssocEnd|
  oneRole.class=oneCM; oneRole.assoc=assoc;
  manyRole.class=manyCM; manyRole.assoc=assoc;
  oneRole.multiplicity='1'; manyRole.multplc='*';
  oneRole.roleName<>oclUndefined(String);
  manyRole.roleName<>oclUndefined(String);}
 check enforce dm(oneDM:ClassDM;manyDM:ClassDM){
  realize oneAttr:Attr, realize manyAttr:Attr,
  realize oneOp:Op, realize manyOp:Op |
  oneAttr.class:=manyDM; manyAttr.class:=oneDM;
  oneOp.class:=manyDM; manyOp.class:=oneDM;
  oneAttr.type:=oneDM.name;
  oneOp.retType:=oneDM.name;
  manyAttr.type:=
        'Set('.concat(manyDM.name).concat(')');
  manyOp.retType:=manyAttr.type;}
 where(oneMapping:Cm2Dm, manyMapping:Cm2Dm){
  realize one2Attr:AssocEnd2Attr,
  realize many2Attr:AssocEnd2Attr |
  one2Attr.role:=oneRole;one2Attr.attr:=oneAttr;
  one2Attr.op:=oneOp; many2Attr.role:=manyRole;
  many2Attr.attr:=manyAttr;many2Attr.op:=manyOp;
  oneAttr.name:=oneRole.roleName;
  oneOp.name:='get'.concat(oneRole.roleName);
  manyAttr.name:=manyRole.roleName;
  manyOp.name:='get'.concat(manyRole.roleName)}}
```

Fig. 3. Example Transformation in the QVT Core language

and two *Op* objects are created and their attribute values are updated by the assignments in the bottom part. The keyword '`realize`' means that the realized variables must be enforced in order to bind to values. The keyword '`enforce`' means that in case the OCL constraints within the bottom pattern do not hold, the target model, i.e., the design model must be changed to fulfil the constraints.

The transformation executes when its mappings are executed. A combination of the two mappings *cm2Dm* and *assocEnd2Attr* fulfils the requirements stated for the example transformation *addAssociation*.

2.2 TGGs Including OCL for the Example Transformation

Figure 4 presents the rule *addAssociation* in a TGG-like style, but in addition to TGG elements OCL is also used. In the six parts LG, CG, RG, LB, CB, and RB, the items L, C, R, G, B stand for Left, Correspondence, Right, Guard and Bottom, respectively. The notions Guard and Bottom are taken from the QVT Core language. The properties in the lower part of the object rectangles (like *name* or *multiplicity*) indicate object attributes. The TGG rule is a non-deleting rule, i.e., the left-hand side (L=LG+CG+RG) is included in the right-hand side (R=L+LB+CB+RB).

In Fig. 4, OCL constraints for the rule are presented. The OCL constraints cannot only be employed to restrict the attribute values in the guard and bottom part, but they can also express more general restrictions. Classical TGG rules do not embody OCL constraints. Our approach allows to add OCL constraints in all parts and thus enables a general form for restricting the rule application as well

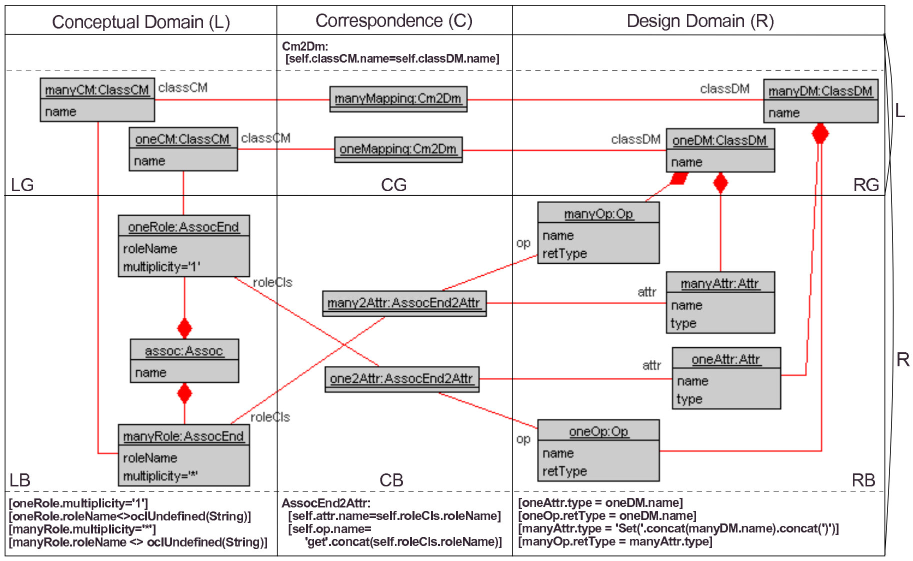


Fig. 4. Example TGG rule including OCL: *addAssociation*

as a fine-grained description of consequences of rule application, for example, to describe the updating of object attributes. General OCL expressions, which can traverse the complete working graph on which the rule is applied, are allowed as right sides of attribute assignments. The evaluation of the OCL expressions is made before the working graph is modified by the rule.

This rule works as follows: Applying the rule within the conceptual domain means to create an *Assoc* object representing an association; this induces a graph rewrite in the design domain, i.e., *Attr* objects and *Op* objects representing the association are created; the correspondence between the added association and the new attributes and operations is established by a graph rewrite in the correspondence domain. The OCL constraints allow to check the applicability of the rule and to update the attribute values within the design domain.

2.3 Requirements for the Integration of TGGs and OCL

By considering QVT as a motivation and a target for the integration of TGGs and OCL as well as by generalizing the results from the subsections 2.1 and 2.2, we propose requirements for the integration of TGGs and OCL as follows.

– Supporting OCL formulas for attribute expressions.
– Supporting OCL conditions as pre- and postconditions of TGG rules (corresponding to negative application conditions and the right-hand side of TGG rules, respectively), which cannot be represented by graphs.

– Supporting constraints on the metamodel of the middle domain of TGG rules. (Supposing that all of the constraints must be taken from TGG rules)
– Supporting bi-directionally executable TGG rules in a synchronous way.

In addition, the integration of TGGs and OCL requires variations of TGGs. A node in the correspondence domain may be connected to multiple nodes in the remaining domains. This is a variation of TGGs because in the original TGGs [1], the mappings are only one-one mappings. With the integration, TGG rules can be deleting rules whereas in the definition of TGGs [1], TGG rules are non-deleting rules.

3 USE Realization of TGGs and OCL

This section presents an approach for the realization of TGG rules including OCL within our tool USE. We propose a language called *use4tgg* for the integration of TGGs and OCL. We obtain operational scenarios of a TGG rule by automatically translating the *use4tgg* description of the TGG rule. The operations are realized in two ways: OCL pre- and postconditions are employed as operation contracts, and command sequences are taken as an operational and executable realization. We extend the technique from previous work [10] in order to realize operational scenarios of a TGG rule.

3.1 Descriptions in the Language use4tgg

Figure 5 presents a *use4tgg* specification for the TGG rule *addAssociation*. Recall that the rule is also represented in Fig. 3 and Fig. 4.

A TGG rule in *use4tgg* includes six parts. Each part corresponds to a pattern of the TGG rule. Each pair of them corresponds to a domain (left, correspondence, right) rule: (LG, LB), (CG, CB) and (RG, RB). As above, G stands for Guard and B for Bottom. A rule within a domain is specified by the guard and bottom parts (*G, *B). They can be seen as the pre- and postcondition of the rule, respectively.

The language *use4tgg* offers a particular feature for the correspondence domain and allows an extension for specifying patterns of the correspondence. In use4tgg one can specify correspondences between objects of the left and right domains. The syntax for these so-called correspondence links is as follows.

(object_L, object_R) as (role_L, role_R) in object_C: Class_C

The items L, R, and C stand for Left, Right and Correspondence, respectively. The advantage of this extension is that we can automatically generate the metamodel of the correspondence domain. For example, such a correspondence link is employed in the (CG, CB) parts in Fig. 5. From these correspondence links, we can derive the metamodel of the correspondence domain in Fig. 2. In addition, these correspondence links allow to suppress links between the correspondence domain and the left and right domains, i.e., the two links (*object_L*, *object_C*)

```
rule addAssociation
LG---------------------------------------------------------------
  oneCM:ClassCM
  manyCM:ClassCM
LB---------------------------------------------------------------
  assoc:Assoc
  oneRole:AssocEnd
  manyRole:AssocEnd
  (assoc,oneRole):Assoc_AssocEnd
  (assoc,manyRole):Assoc_AssocEnd
  (oneCM,oneRole):Class_AssocEnd
  (manyCM,manyRole):Class_AssocEnd
  [oneRole.multiplicity='1']
  [oneRole.roleName<>oclUndefined(String)]
  [manyRole.multiplicity= '*']
  [manyRole.roleName<>oclUndefined(String)]
RG---------------------------------------------------------------
  oneDM:ClassDM
  manyDM:ClassDM
RB---------------------------------------------------------------
  oneAttr:Attr
  manyAttr:Attr
  oneOp:Op
  manyOp:Op
  (oneDM,manyAttr):Class_Attr
  (manyDM,oneAttr):Class_Attr
  (oneDM,manyOp):Class_Op
  (manyDM,oneOp):Class_Op
  [oneAttr.type=oneDM.name]
  [oneOp.retType=oneDM.name]
  [manyAttr.type='Set('.concat(manyDM.name).concat(')')]
  [manyOp.retType=manyAttr.type]
CG----------------------------------------------------------------
  (oneCM,oneDM) as (classCM,classDM) in oneMapping:Cm2Dm
  (manyCM,manyDM) in manyMapping:Cm2Dm
  Cm2Dm:[self.classCM.name=self.classDM.name]
CB---------------------------------------------------------------
  (oneRole,oneAttr) as (roleCls,attr) in one2Attr:AssocEnd2Attr
  (oneRole,oneOp) as (roleCls,op) in one2Attr:AssocEnd2Attr
  (manyRole,manyAttr) in many2Attr:AssocEnd2Attr
  (manyRole,manyOp) in many2Attr:AssocEnd2Attr
  AssocEnd2Attr:[self.attr.name=self.roleCls.roleName]
  AssocEnd2Attr:[self.op.name='get'.concat(self.roleCls.roleName)]
end
```

Fig. 5. TGG rule *addAssociation* in use4tgg

and (*object_R*,*object_C*) do not have to be given explicitly, because they can be inferred from the correspondence link. Correspondence links highlight the correspondence in a condensed way.

OCL constraints from the correspondence domain (CG, CB) can become constraints (invariants) on the metamodel. For example, OCL constraints in the CG and CB parts in Fig. 5 may be employed for restricting the metamodel in Fig. 2.

3.2 Realization by OCL Pre- and Postconditions

The rule can be realized by OCL pre- and postconditions as an operation contract. We obtain these OCL constraints by applying a sequence of transformation steps on the rule: Initialization, Link Preservation, Link Creation, Link Destruction, Object Creation, Object Preservation and Object Destruction [10].

Figure 6 illustrates the derivation of OCL pre- and postconditions by the transformation steps. The Initialization step is applied on the objects a, b, and d and the links link1 and link2. The Link Preservation step is executed on

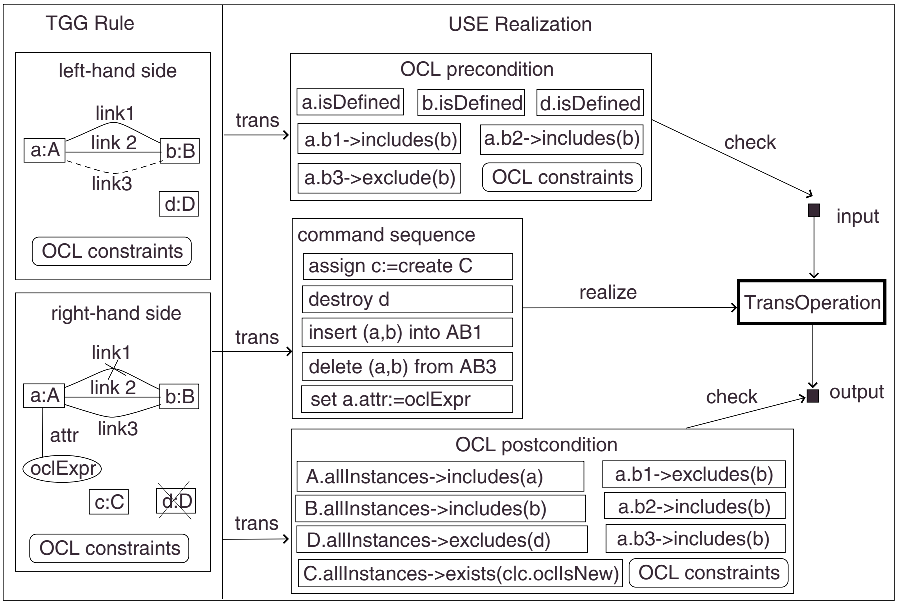


Fig. 6. USE realization for a TGG rule: (i) By imperative command sequences, and (ii) by declarative OCL pre- and postconditions

link2. The Link Creation step creates link3. Then the Link Destruction step is done on link1. The Object Creation step creates the object c. Afterwards, the Object Preservation step is applied on the objects a and b, and the Object Destruction step on the object d. Using these steps, OCL pre- and postcondition are derived. For example, when the Link Destruction step is applied on link1, we obtain the OCL precondition `a.b1→includes(b)` and the OCL postcondition `a.b1→excludes(b)`.

3.3 Realization by Command Sequences

We employ five basic commands: Object Creation, Link Creation, Attribute Assignment, Link Destruction and Object Destruction in order to realize the operation. The commands are illustrated by the part ‘command sequence’ in Fig. 6.

Remark 1: Although we do not force to employ assignments in our patterns, certain OCL formulas must be presented in an assignment-like style, if we want to obtain a complete operation realization of a TGG rule in form of a command sequence: Only if in the bottom part of a TGG rule an attribute modification (written down as an OCL formula) is presented in the form `Attribute` = `OclExpression`, the resulting operation is realized in a correct way.

Remark 2: *use4tgg* has two modes for specifying TGG rules. The first mode is indicated with the keyword NONDELETING and the keywords for the rule parts (LG, CG, RG, LB, CB, RB), and the second mode is indicated with the keyword EXPLICIT. The first mode is employed for specifying non-deleting rules, e.g., for treating the example transformation in this paper. This mode is the default and the keyword NONDELETING may be skipped. The second mode is employed for specifying rules which may be deleting. In this mode, all nodes and links must be explicitly stated whereas in the first case only the added nodes and links are mentioned. The pattern format of the two modes is identical. For specifying non-deleting rules, the first mode is much shorter than the second mode.

3.4 Deploying the TGG and OCL Realization in USE

Figure 7 presents the result of applying the transformation *addAssociation*. Snapshot1 is glued with the guard part of the rule, and Snapshot2 is the result of the rule application. These object diagrams represent the abstract syntax of the example, whereas the diagram in Fig. 1 is another representation in concrete syntax.

The integration of TGGs and OCL is completely covered by USE [8]. This is similar to the case for realizing graph transformation within USE [11].

Presenting models: Host models are represented as object diagrams in USE. The models are restricted by the metamodel invariants.

Matching TGG rules: Matching a rule is carried out by evaluating OCL queries on the source object diagram (working graph). These queries are captured by the precondition of the operation corresponding to the rule.

Applying TGG rules: A rule application is realized by an operation call. After each rule application, one may check the postconditions of the rule for an on-the-fly verification of the transformation.

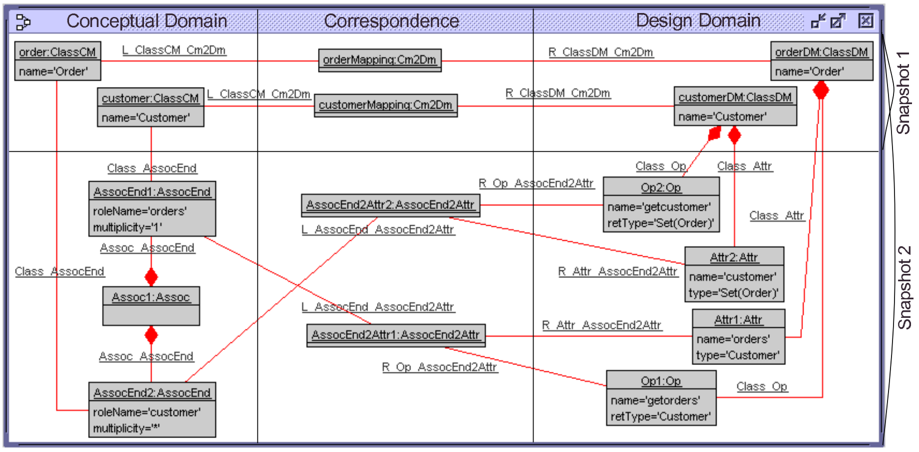


Fig. 7. Applying the TGG rule *addAssociation*: From Snapshot1 to Snapshot2

4 Support Model Co-evolution and Consistency

This section discusses how the integration of TGGs and OCL can support model co-evolution and consistency by means of an example. We want to emphasize again the consistency between a conceptual model and a design model and to maintain the consistency whilst both models are evolving or changed. In exemplary form, Fig. 8 illustrates the situation with the model versions A, B and C. The conceptual (left) and design (right) models are presented in concrete syntax. But the following discussion refers to the models as instances of the UML metamodel.

Co-evolution of models. A co-evolution step between model versions can be represented by a TGG rule including OCL. The left- and right-hand sides of the rule correspond to the target and source versions. The consistency between the models in the different versions is maintained by links and OCL constraints in the correspondence domain of the TGG rule. The co-evolution is carried out by a transformation operation derived from the TGG rule: the source version is transformed to the target version. In Fig. 8, see the note (1), a co-evolution from the version A to the version B is carried out by the TGG rule *transAssocclass_one2many* as the association multiplicity in the conceptual model in the version A is changed. The correspondence between the models in the versions A or B expresses constraints for a refinement of an association class in the conceptual model, e.g., names of corresponding Class (Attribute) objects must coincide. In the version A, we recognize the correspondence between two Attribute objects whose *name* attribute is ‘salary’. The

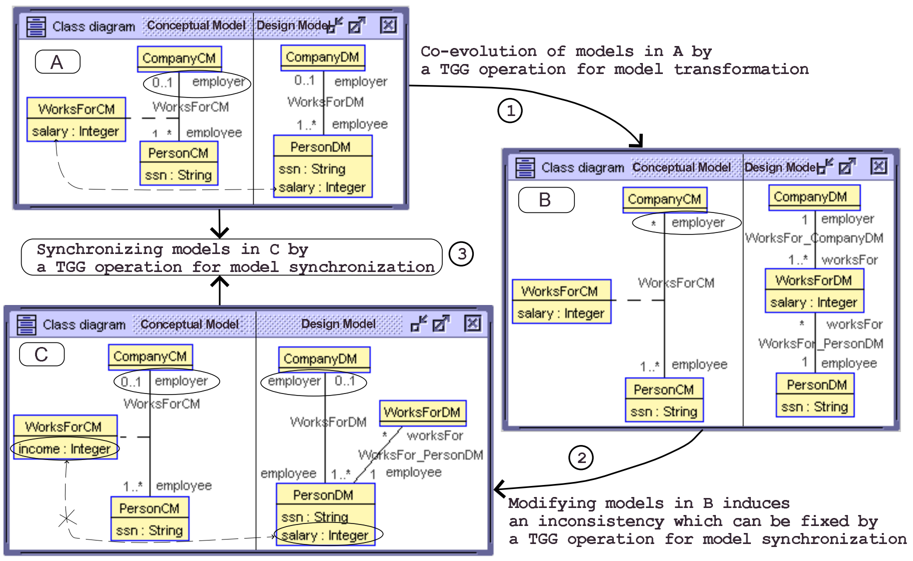


Fig. 8. Model co-evolution and consistency with TGG rules including OCL

correspondence (the dashed line in A) is represented by links between the objects and an object Cm2Dm in the correspondence domain of the TGG rule and the OCL invariant in the class Cm2Dm: `self.attrCM.name=self.attrDM.name`.

Detecting inconsistency. We consider an inconsistency in the target version as (Class, Attribute etc.) object attributes are changed. We can detect the inconsistency by checking OCL constraints in the correspondence domain of the TGG rule. In Fig. 8, see the note (2), the version C results from a modification in the version B. In the version C, we recognize that there are two corresponding Attribute objects whose *name* attributes do not coincide because the value ‘income’ differs from ’salary’. This induces an inconsistency (indicated by the dashed line in C) because the invariant of the class Cm2Dm is not fulfilled: `self.attrCM.name=self.attrDM.name`.

We consider another inconsistency in the target version as links are changed. We can detect the inconsistency in a semi-automatic way by representing models as the image of the TGG rule. We query them from the host models using OCL conditions built by links and OCL constraints in the TGG rule. In Fig. 8, the models in the version C can be queried using OCL conditions derived from the TGG rule *transAssocclass_one2many*. In the version C, we can recognize an inconsistency since object links are changed inducing that the refinement of the association class is not fulfilled.

Fixing inconsistency. Detecting inconsistency in the way as pointed out above allows us to fix the inconsistency manually. We propose an approach for detecting and fixing automatically the inconsistency in the target version. The inconsistency occurs because of the change of (Class, Attribute etc.) object attributes and links in models of the version. We employ a synchronization scenario of the TGG rule in order to fix the inconsistency. The general aim of the operation is to re-transform the source version to the target version, however, instead of creating objects and links like the co-evolution scenario does, the operation will update objects and links in the target version. In Fig. 8, see the note (3), the versions A and C correspond to the left- and right-hand sides of the TGG rule. The synchronization of the version C allows us to fix the inconsistency. A deeper discussion of our realization of the operation goes beyond the scope of the paper.

5 Related Work

Triple Graph Grammars (TGGs) have been proposed first in [1]. Since then, a lot of further developments and applications have indicated the strength of TGGs in particular within software engineering.

In [12], a comparison between TGGs and the QVT Core language is presented. The paper shows that both approaches are very similar. A difference between them is the use of OCL constraints within the QVT Core language whereas multiple correspondence nodes would do the same within TGG. That paper rises the question how to integrate OCL into TGGs. This can be seen as one motivation for our work.

In [6], a quite complete survey of principles, concepts, usability and implementations of TGGs is discussed. This report proposes a notation for OCL constraints within TGG descriptions. However, OCL constraints within that approach are only for attribute expressions. That paper considers only non-deleting rules and does not go into details about the merits of OCL constraints on metamodels as well as the technical realization for OCL.

The approach in [13] considers the integration of OCL into model transformations within Fujaba. This is realized by integrating the Dresden OCL Toolkit. However, that paper does not mention explicitly TGG concepts as well as OCL constraints in connection with TGGs.

The work in [3] proposes an approach for the integration of a number of MOF-compliant models. So-called Multi-Domain Integration (MDI) rules, a generalization of TGGs, are proposed. The paper mentions the value of declarative specifications in connection with TGGs and OCL. However, the specification of OCL constraints within TGGs is not treated.

In [5,4,14], an extension of TGGs for model integration is proposed. The basic idea is that declarative TGG rules may be (semi-)automatically derived in order to obtain operational graph grammar rules for consistency checking, consistency recovery and model transformation. However, within these works, OCL constraints, which are in our view central for such approaches, are not treated.

Our work can be seen an approach to model traceability [15]. The TGG approach supports an interesting feature for the motivation: Trace links can be automatically generated. In [16], the approach to tracing two models by a middle model, a so-called weaving model, is considered. Weaving models are built by weaving metamodels. We can see TGGs as the very transformations on weaving models. OCL constraints within our integration of TGGs and OCL are the very constraints for correspondences within weaving models.

The work in [17] outlines an approach utilizing TGGs for managing consistency between views in general and UML models and semantic domains in particular. In [7], TGGs are employed for model view management. [18] presents a TGG approach for incremental model synchronization. Our work adapts from these papers the intended application scenarios.

Plain textual OCL has been extended to be visualized with collaborations in [19]. Within our previous work [11,10], we proposed an approach for the realization of plain graph transformation in USE. The current work extends this approach for treating TGGs.

6 Conclusion and Future Work

This paper proposes a language for the integration of Triple Graph Grammars (TGGs) and the Object Constraint Language (OCL). The approach views operational scenarios of a TGG rule as operations and characterizes the operations by OCL pre- and postconditions and by an executable command sequence. The language is realized in our UML and OCL tool USE. USE supports full

OCL and UML (metamodel) class diagrams. Thus, metamodels and their transformations can be inspected and analysed in USE. This paper proposes a novel approach for model co-evolution and consistency using the integration of TGGs and OCL.

Future work will include a graphical user interface in combination to the current textual interface for the specification of rules. We aim to extend our language to special QVT features expressed in QVT by keywords as 'check', 'enforce' and 'realize'. A further activity will concentrate on how to generate correspondence metamodels from TGG rules. Middle and large scale case studies must give feedback on the practical applicability of our work. In particular, we see applications in the fundamentals of language engineering where a source language (including syntax and semantics) is translated into a target language (also including syntax and semantics) and where the correspondence domain and the correspondence metamodels take care of correctness criteria.

References

1. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer, Heidelberg (1995)
2. Pratt, T.W.: Pair Grammars, Graph Languages and String-to-Graph Translations. Academic Press 5, 560–595 (1971)
3. Königs, A., Schürr, A., Bézivin, J., Heckel, R.: Multi-Domain Integration with MOF and extended Triple Graph Grammars. In: Language Engineering for Model-Driven Software Development, IBFI, Schloss Dagstuhl, Germany, vol. 04101 (2005)
4. Königs, A.: Model Transformation with Triple Graph Grammars. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713. Springer, Heidelberg (2005)
5. Königs, A., Schürr, A.: Tool Integration with Triple Graph Grammars - A Survey. ENTCS 148, 113–150 (2006)
6. Kindler, E., Wagner, R.: Triple Graph Grammars: Concepts, Extensions, Implementations and Application Scenarios. Technical Report, University of Paderborn, Germany (2007)
7. Guerra, E., de Lara, J.: Model View Management with Triple Graph Transformation Systems. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 351–366. Springer, Heidelberg (2006)
8. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming (2007)
9. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Final Adopted Specification ptc/07-07-07. OMG (2007)
10. Büttner, F., Gogolla, M.: Realizing Graph Transformations by Pre- and Postconditions and Command Sequences. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 398–413. Springer, Heidelberg (2006)
11. Gogolla, M., Büttner, F., Dang, D.H.: From Graph Transformation to OCL using USE. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) AGTIVE 2007. LNCS, vol. 5088, pp. 585–586. Springer, Heidelberg (2007)
12. Greenyer, J., Kindler, E.: Reconciling tGGs with QVT. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 16–30. Springer, Heidelberg (2007)

13. Stölzel, M., Zschaler, S., Geiger, L.: Integrating OCL and Model Transformations in Fujaba. In: Chiorean, D., Demuth, B., Gogolla, M., Warmer, J. (eds.) Models/UML Workshop on OCL (OCLApps 2006), ECEASST, vol. 5 (2006)
14. Grunske, L., Geiger, L., Lawley, M.: A Graphical Specification of Model Transformations with Triple Graph Grammars. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 284–298. Springer, Heidelberg (2005)
15. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model Traceability. IBM Systems Journal 45 (2006)
16. Fabro, M.D.D., Valduriez, P.: Semi-automatic Model Integration Using Matching Transformations and Weaving Models. In: Cho, Y., Wainwright, R.L., Haddad, H., Shin, S.Y., Koo, Y.W. (eds.) SAC, pp. 963–970. ACM, New York (2007)
17. Heckel, R., Küster, J., Taentzer, G.: Towards Automatic Translation of UML Models into Semantic Domains. In: Kreowski, H.J., Knirsch, P. (eds.) AGT Workshop on Applied Graph Transformation, Grenoble, France, pp. 11–22 (2002)
18. Giese, H., Wagner, R.: Incremental Model Synchronization with Triple Graph Grammars. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 543–557. Springer, Heidelberg (2006)
19. Bottoni, P., Koch, M., Parisi-Presicce, F., Taentzer, G.: A visualization of OCL using collaborations. In: Gogolla, M., Kobryn, C. (eds.) UML 2001. LNCS, vol. 2185, pp. 257–271. Springer, Heidelberg (2001)

Triple Graph Grammars or Triple Graph Transformation Systems?
A Case Study from Software Configuration Management

Thomas Buchmann, Alexander Dotor, and Bernhard Westfechtel

Angewandte Informatik 1, Universität Bayreuth
D-95440 Bayreuth
firstname.lastname@uni-bayreuth.de

Abstract. Triple graph grammars have been used to specify consistency maintenance between inter-dependent and evolving models at a high level of abstraction. On a lower level, consistency maintenance may be specified by a triple graph transformation system, which takes care of all operational details required for executing consistency maintenance operations. We present a case study from software configuration management in which we decided to hand-craft a triple graph transformation system rather than to generate it from a triple graph grammar. The case study demonstrates some limitations concerning the kinds of consistency maintenance problems which can be handled by triple graph grammars.

1 Introduction

Model transformations play a key role in model-driven engineering. In the most simple case, a transformation of some source model s into some target model t may be performed automatically by a model compiler. If there is no need to edit t, model evolution (of s) may be handled by simply compiling s once more. However, in general it may not be possible to generate t from s completely automatically, both s and t may evolve, and changes may have to be propagated in both directions (round-trip engineering).

Many formalisms have been proposed for defining model transformations, including e.g. QVT [1] in the context of object-oriented modeling. In this paper, we focus on *graph transformations*: Graphs may represent models in a natural way; graph transformation rules describe modifications of graph structures in a declarative way. Furthermore, there is a comprehensive body of theories, languages, tools, and applications (see e.g. the handbooks on graph grammars [2,3]).

A *directed, typed, attributed graph* consists of typed nodes which are decorated with attributes and are connected by typed, directed, binary edges. In terms of object-oriented modeling, a node corresponds to an object, and an edge corresponds to a binary link with distinguishable ends. A *graph grammar* is composed of a graph schema, which defines types of nodes, edges, and attributes, a start graph, and a set of *productions* which are used to generate a set of graphs forming a graph language. Thus, a graph grammar is concerned only with the generation of graphs. In contrast, a *graph transformation system* is made up of a set of *graph transformation rules*, which describe arbitrary graph transformations including deletions and modifications.

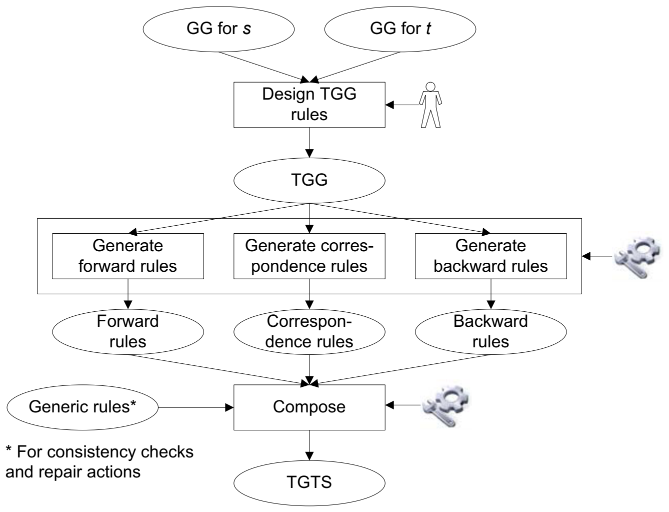


Fig. 1. The TGG approach

For maintaining consistency between interdependent and evolving models, a graph transformation system is required which deals with at least two graphs, namely the *source graph* s and the *target graph* t. For incremental change propagation, a *correspondence graph* c is placed in between s and t for maintaining *traceability links* with the help of *link nodes*. This results in a *triple graph transformation system (TGTS)*, the rules of which define source-to-target and target-to-source transformations as well as actions for checking consistency and repairing inconsistencies.

Developing a TGTS is still a tedious and laborious task: First, in addition to the generation of graphs, modifications and deletions have to be specified explicitly. Second, each direction of transformation has to be specified explicitly - as well as rules for checking consistency and establishing correspondences. Therefore, *triple graph grammars (TGG)* [4,5] have been proposed to leverage the specification of inter-model consistency maintenance. From a high-level TGG dealing only with the synchronous generation of graphs, a more low-level TGTS may be generated (Figure 1): The TGG designer need only define generative *triple rules* which describe synchronous extensions of source, target, and correspondence graph. From each synchronous triple rule, three *directed rules* may be generated (if required): A *forward rule* assumes that s has been extended, and extends c and t accordingly; likewise for *backward rules*. A *correspondence rule* extends c after s and t have been extended with corresponding graph structures. Like triple rules, directed rules are *monotonic*, i.e., they describe graph extensions. Directed rules are required when s, t, and c have not been changed synchronously (e.g., when different users edit s and t independently). In addition to directed rules, further rules are needed which deal with modifications and deletions having been performed in s and t. To this end, *generic rules* have to be defined which perform *consistency checks* and *repair actions*. Finally, the TGTS may include a (generic) control structure for efficient graph traversal to speed up the execution of consistency maintenance operations.

In this paper, we explore the alternatives of *hand-crafting a TGTS* versus defining a TGG and *generating a TGTS* from the TGG. To this end, we present a *case study* which we performed in the *MOD2-SCM* project (*MOD*el-Driven *MOD*ular *S*oftware *C*onfiguration *M*anagement System [6]), which aims at developing a model-driven product line for SCM systems. The project employs Fujaba [7] as modeling language and tool, but the discussion in this paper is not specific to Fujaba.

2 Case Study

In the context of developing SCM systems, a recurring problem to be solved consists in the *synchronization* between *repositories* and *workspaces*, which requires *bidirectional* and *incremental change propagation*. The case study was inspired by open source SCM systems such as Subversion or CVS. In the model we built, both files and directories are versioned in a uniform way. The version history of each file system object is represented by a *version tree*. A version of a directory uniquely determines the versions of its components. While file system objects are organized into strict hierarchies (trees), the hierarchy of file system object versions allows for sharing.

Synchronization between repositories and workspaces is supported by the following commands which, when applied to directories, are propagated to all components: add prepares a file system object for a commit into the repository. commit commits a file system object into the repository. For an added file system object, an initial version is created in the repository. For a changed versioned file system object, a successor version is created. checkout creates a fresh copy of some file system object version in the workspace. update propagates changes from the repository into the workspace. Unmodified copies of outdated versions are replaced with copies of new versions; modified copies are not updated to prevent the loss of local changes. Finally, checkStatus analyzes the consistency between repository and workspace. File system objects are marked as created, deleted, modified, moved, or out of date.

An example is given in Figure 2. The scenario starts with a repository which is initially empty and a small file system hierarchy (a). In step 1, add is applied to the hierarchy rooted at d1. All objects (files or directories) of this hierarchy are scheduled for commit. Adding is implemented by creating *link objects* and attaching them to the file system objects. In step 2, commit is used to import the file hierarchy into the repository. For each file system object, both a versioned object and its initial version are created. Furthermore, the hierarchy is mirrored in the repository both at the object and the version level. In step 3, a part of the hierarchy (with root d2) is exported into another workspace with checkout. In ws2, the text content of f2 is updated, and the name of f3 is changed into f4. Finally, in step 4 the changes in ws2 are committed. This causes new versions of both files to be created. Please note that the file names of different versions may differ. Furthermore, a new version of the directory d2 is created which includes the new versions of the changed files. An update propagates the changes from the repository to ws1; all files and directories now refer to the current versions. Please note that due to the lack of space we have not shown *structural variations* caused by moves and deletes.

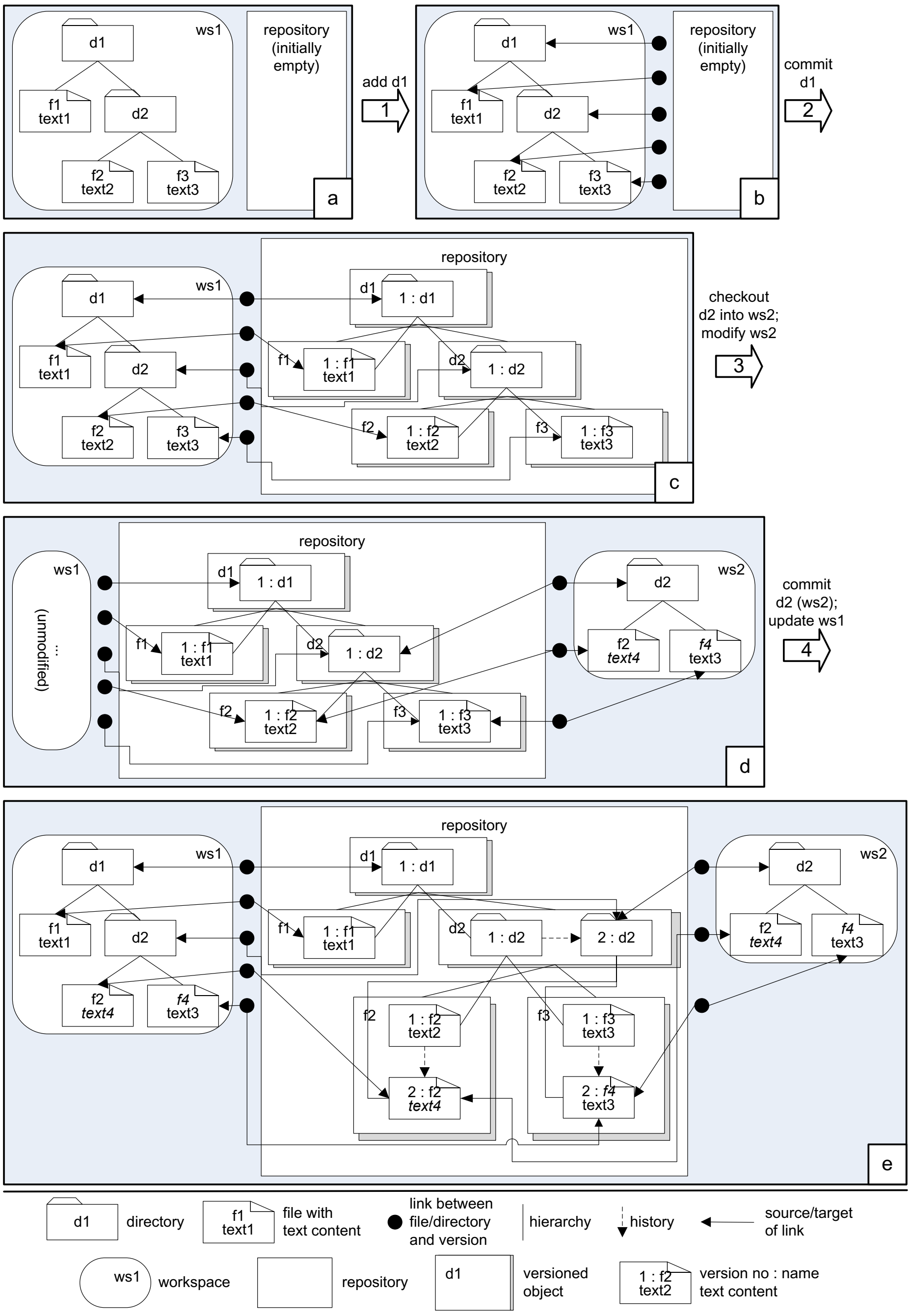


Fig. 2. Example

3 A Triple Graph Transformation System

In this section, we present the approach which we decided to follow in our case study. Synchronization of repositories and workspaces is modeled with the help of a *hand-crafted TGTS*. Generating a TGTS from a TGG is discussed in the next section.

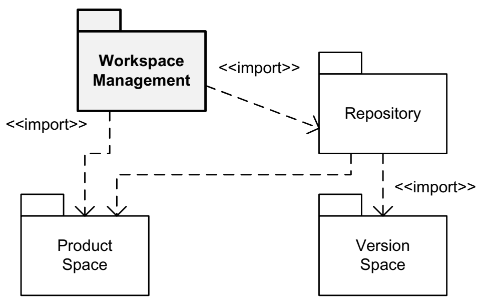
Modeling Language. The TGTS was created with the help of the object-oriented modeling language and environment *Fujaba* [7]. In Fujaba, nodes and edges are represented as objects and links, respectively. Types of nodes and edges are defined by *class diagrams*. The behavior of objects and links may be modeled by story patterns and story diagrams (see below). Models written in Fujaba are executable (compiled into Java code).

Story patterns are UML-like communication diagrams which can be used to represent graph transformation rules. A story pattern consists of a graph of objects and links. A graph transformation rule is expressed by marking graph elements as deleted or created. Furthermore, conditions may be defined for attribute values, and the values of attributes may be modified. Finally, methods may be called on objects of the story patterns. Thus, story patterns constitute a uniform language construct for defining graph queries, graph transformations, and method calls.

Programming with story patterns is supported by *story diagrams*, which correspond to interaction overview diagrams in UML 2.0. A story diagram consists of story patterns which are arranged into a control flow graph. Story patterns appear in-line in nodes of the control flow graph. Each story diagram is used to implement a method defined in the class diagram.

Model Architecture. Figure 3 shows a *package diagram* on the left and lists data on the overall model size on the right. Each *package* contains one class diagram and a set of story diagrams for the methods introduced in the classes of the package. A dashed line labeled with import denotes an *import* relationship. An unlabeled dashed line represents a *dependency* (the dependent package may have to be modified if the target package is changed).

Using the terminology of [8], we distinguish between a *product space*, which is composed of the elements to be submitted to version control, and a *version space*, where



number of classes and interfaces	24
number of attributes	27
number of methods	83
lines of generated Java code	13291

Fig. 3. Package diagram (left) and model size (right)

the evolution of these elements is represented. In the context of this paper, the product space consists of the file system hierarchy. The package VersionSpace introduces classes for managing the evolution of versioned objects. In the version model of the case study, the versions of a versioned object are organized into a history tree. The package Repository provides classes for versioned files and directories, based on the packages ProductSpace and VersionSpace. Composition hierarchies are defined on both object and version level. The packages mentioned so far will not be discussed further. Rather, we will focus on the package WorkspaceManagement.

Class diagram. Figure 4 shows the class diagram for the package Workspace Management. Classes imported from other packages are displayed with collapsed attributes and methods sections (left-hand and right-hand side). The class WorkspaceManager provides the external interface of methods for synchronizing repositories and workspaces (facade pattern).

Synchronization between repositories and workspaces is realized with the help of a *correspondence graph*, which is composed of *link objects* being connected to one source object and one target version, respectively. The abstract class Link provides a set of methods corresponding to the exported methods of WorkspaceManager, and a set of auxiliary methods. Furthermore, each link object carries a set of boolean attributes indicating the state of the link: In state created, the target of the link does not yet exist. In state deleted, the source of the link has been destroyed. In state modified, both source and target exist, and the source has been modified in the workspace. In state moved, the source has been moved to a different location. In state outOfDate, a new version of the target is available. Finally, in state updated, the source has been updated to a new version of the target, but this change has not been committed yet at the next higher level. Please note that these attributes are not mutually exclusive (e.g., a link object can be both out of date and modified).

Link objects are organized into a composition hierarchy in a similar way as file system objects (composite pattern). When the workspace is consistent with the repository, the composition tree for link objects agrees with the composition tree of file system objects in the workspace. The algorithms for synchronizing repositories and workspaces traverse the composition hierarchy. Since they are attached to the class Link and its subclasses, the classes of the imported packages need not be extended or modified.

Operations. To illustrate the style of modeling adopted in the TGTS, we present three simple examples of methods for synchronizing repositories and workspaces. All sample methods are attached to the class Link and perform only those parts of the synchronization which can be handled at this general level. The methods are redefined in the subclasses; propagation through the hierarchy is performed in the class CompositeLink. Only link objects and their embeddings are manipulated directly. All changes in the repository and the workspace are affected by method calls only.

Figure 5a shows the story diagram for committing the creation of a new file system object. The first story pattern checks the state of the link object and locates the root of the repository via the workspace manager. In the second step, a new versioned object is created. The third step creates the root version of this object, sets its name and its owner, and connects the new version to the link object.

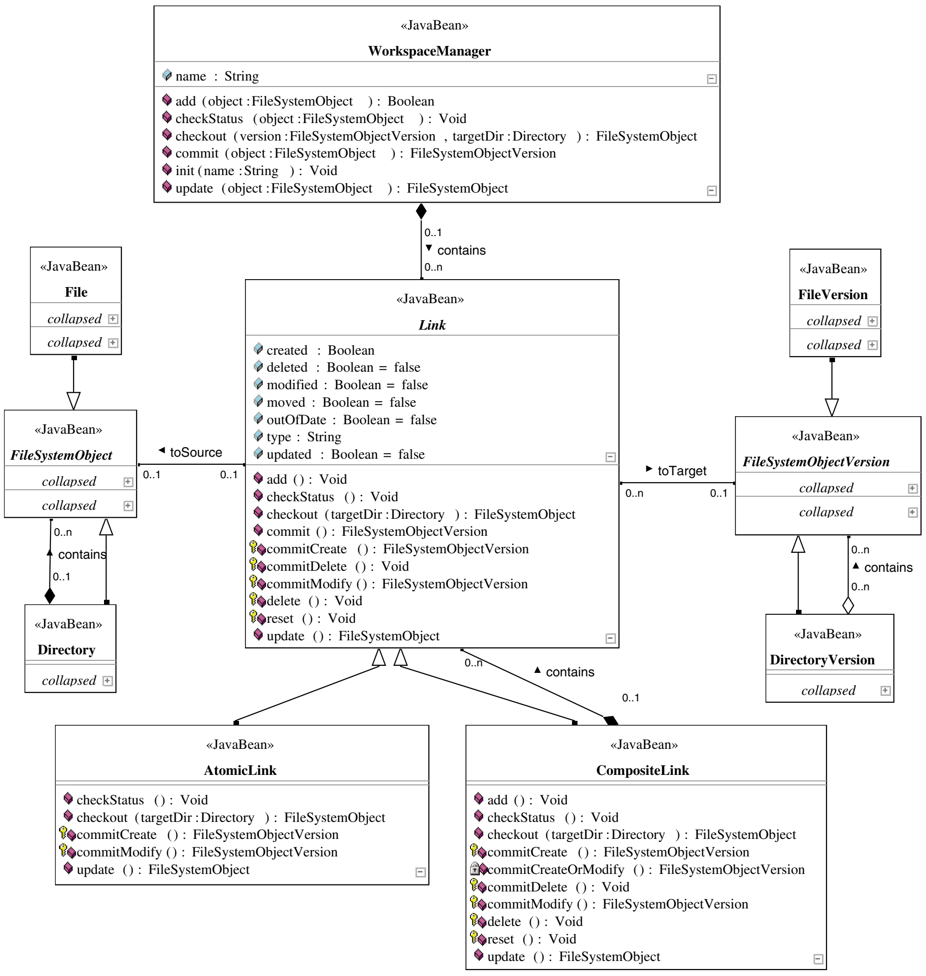

«JavaBean»
WorkspaceManager
name : String
add (object : FileSystemObject) : Boolean
checkStatus (object : FileSystemObject) : Void
checkout (version : FileSystemObjectVersion , targetDir : Directory) : FileSystemObject
commit (object : FileSystemObject) : FileSystemObjectVersion
init (name : String) : Void
update (object : FileSystemObject) : FileSystemObject
0..1
contains
0..n
«JavaBean»
Link
created : Boolean
deleted : Boolean = false
modified : Boolean = false
moved : Boolean = false
outOfDate : Boolean = false
type : String
updated : Boolean = false
add () : Void
checkStatus () : Void
checkout (targetDir : Directory) : FileSystemObject
commit () : FileSystemObjectVersion
commitCreate () : FileSystemObjectVersion
commitDelete () : Void
commitModify () : FileSystemObjectVersion
delete () : Void
reset () : Void
update () : FileSystemObject
«JavaBean»
File
collapsed
collapsed
«JavaBean»
FileSystemObject
collapsed
collapsed
toSource
0..1
0..1
0..n
contains
0..1
«JavaBean»
Directory
collapsed
toTarget
0..n
0..1
«JavaBean»
FileVersion
collapsed
collapsed
«JavaBean»
FileSystemObjectVersion
collapsed
collapsed
0..n
contains
0..n
«JavaBean»
DirectoryVersion
collapsed
0..n
contains
0..1
«JavaBean»
AtomicLink
checkStatus () : Void
checkout (targetDir : Directory) : FileSystemObject
commitCreate () : FileSystemObjectVersion
commitModify () : FileSystemObjectVersion
update () : FileSystemObject
«JavaBean»
CompositeLink
add () : Void
checkStatus () : Void
checkout (targetDir : Directory) : FileSystemObject
commitCreate () : FileSystemObjectVersion
commitCreateOrModify () : FileSystemObjectVersion
commitDelete () : Void
commitModify () : FileSystemObjectVersion
delete () : Void
reset () : Void
update () : FileSystemObject

Fig. 4. Class diagram for the package Workspace Management

The story diagram of Figure 5b is invoked when a file system object is already under version control and has been modified in the workspace. In the first step, the state of the link object is checked, and both ends are retrieved. In the second step, a successor version of the old target is created, and the link object is redirected to the new target.

The story pattern of Figure 5c handles change propagation from the repository into the workspace. The link object must be both outOfDate and not modified; the latter condition prevents that changes in the workspace are overwritten inadvertently. The link object is redirected to refer to the most recent (transitive) successor of the old target. This is ensured by the negative application condition (crossed object): There must be no other successor with a higher version number. In the course of the update, the name and the owner of the file system object are copied from the new target version.

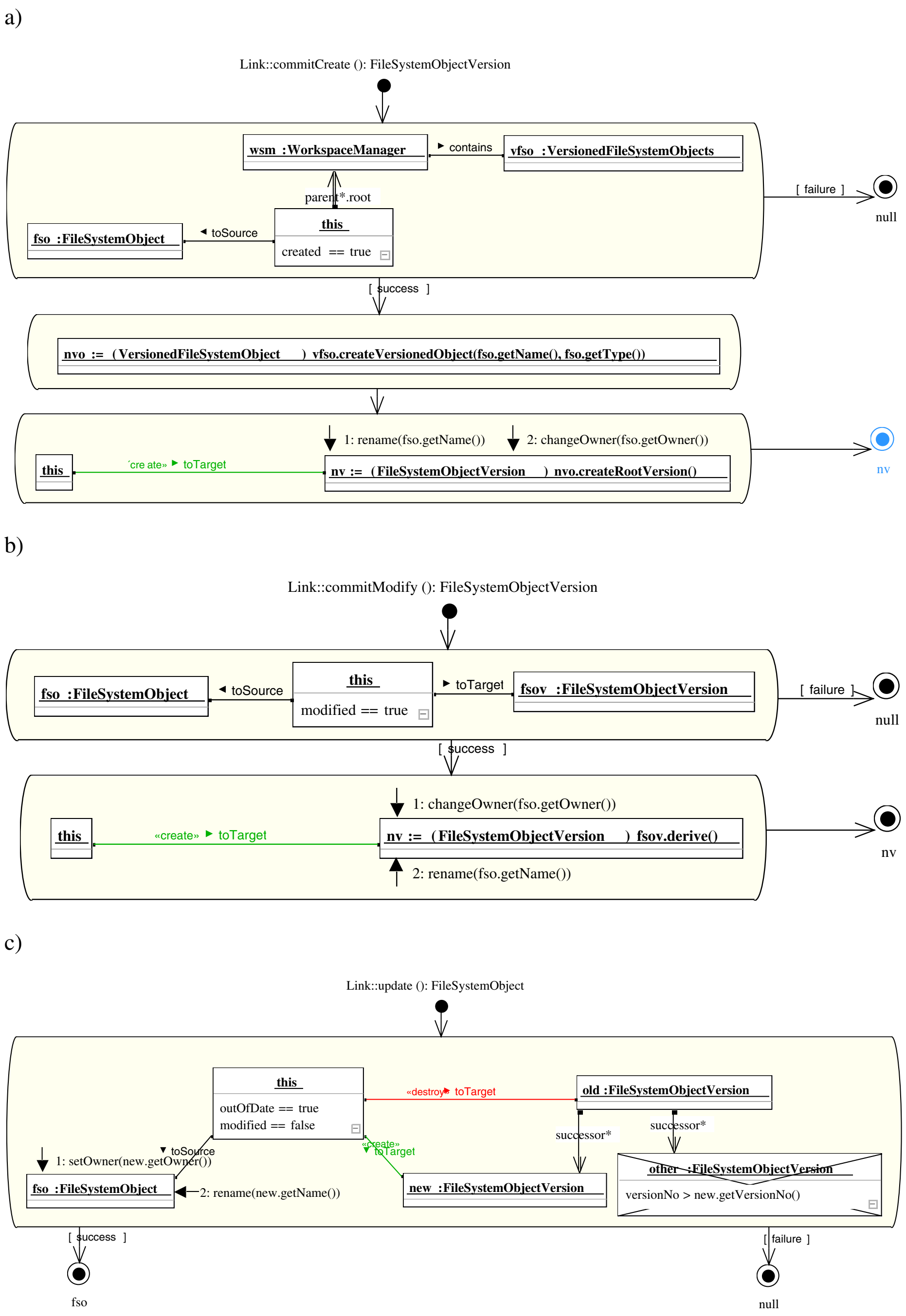

a)
Link::commitCreate (): FileSystemObjectVersion
wsm :WorkspaceManager
contains
vfso :VersionedFileSystemObjects
parent*.root
[failure]
null
this
created == true
fso :FileSystemObject
toSource
[success]
nvo := (VersionedFileSystemObject) vfso.createVersionedObject(fso.getName(), fso.getType())
1: rename(fso.getName())
2: changeOwner(fso.getOwner())
this
«create» toTarget
nv := (FileSystemObjectVersion) nvo.createRootVersion()
nv
b)
Link::commitModify (): FileSystemObjectVersion
fso :FileSystemObject
toSource
this
modified == true
toTarget
fsov :FileSystemObjectVersion
[failure]
null
[success]
1: changeOwner(fso.getOwner())
this
«create» toTarget
nv := (FileSystemObjectVersion) fsov.derive()
2: rename(fso.getName())
nv
c)
Link::update (): FileSystemObject
this
outOfDate == true
modified == false
«destroy» toTarget
old :FileSystemObjectVersion
successor*
successor*
toSource
«create» toTarget
1: setOwner(new.getOwner())
other :FileSystemObjectVersion
versionNo > new.getVersionNo()
fso :FileSystemObject
2: rename(new.getName())
new :FileSystemObjectVersion
[success]
[failure]
fso
null

Fig. 5. Story diagrams for commits and updates

4 A Triple Graph Grammar?

This section discusses the application of the TGG approach to our case study. To this end, we modeled some sample TGG rules in Fujaba. In the presentation below, we describe several conceptual problems which we encountered.

Synchronous Rules. The first step of the TGG approach — defining a set of triple rules — is illustrated in Figure 6. The figure displays three rules which handle the creation of a subdirectory. Each rule is applied to a complex link both ends of which are already present. All rules perform the same extensions on the source graph and the correspondence graph: A subdirectory is created if there is no file system object of the same name; furthermore, a sublink is created and connected to the new subdirectory. The rules differ with respect to the target graph. The rule set is designed in such a way that all structures which may occur in the repository can be generated (version trees for the evolution history, acyclic graphs for the composition hierarchy).

The first rule creates a directory in the workspace along with a versioned object and an initial version in the workspace. The attribute nextVersionNo is a counter which stores the next available version number at the versioned object. The second rule creates a successor version, which is assigned the next available version number, and increments the counter. Using only the first and the second rule, only composition trees may be created in the repository. In the third rule, a directory is created in the workspace and related to a reused version which is added to the version of the parent directory.

Unfortunately, the third rule (CreateDirectoryAndReuseVersion) does not operate correctly. After its execution, the composition hierarchy rooted at the new directory in the workspace is empty, but this does not necessarily apply to the reused version at the other end of the link. If the composition hierarchy below the reused version is not empty, the generation process will "get of out sync". This problem could be fixed by adding directed rules which copy the composition hierarchy into the workspace, but this would break the TGG paradigm (relying on synchronous rules only).

The style of modeling employed in the TGG is quite different from the style of the TGTS. The TGG consists of productions which are partially obtained by copying productions from the grammars for the source and target graphs, respectively (*white-box reuse*). In contrast, in the TGTS source and target graph may be read, but they may be updated only through method calls. This *grey-box reuse* avoids duplication of consistency checks and transformations.

Directed Rules. While synchronization of repositories and workspaces involves bidirectional change propagation with forward and backward rules, *correspondence rules* are of little use: An analysis tool which discovers correspondences between repositories and workspaces and extends the correspondence graph accordingly is not required. The status check, which analyzes consistency between repository and workspace, merely determines the status of already existing link objects (see next paragraph).

Forward rules are obtained from synchronous triple rules by converting created elements in the source graph into elements of the left-hand side. For the rules of Figure 6, this means that the directory nd and its composition link have to be moved to the left-hand side (i.e., they must already be present to apply the rule).

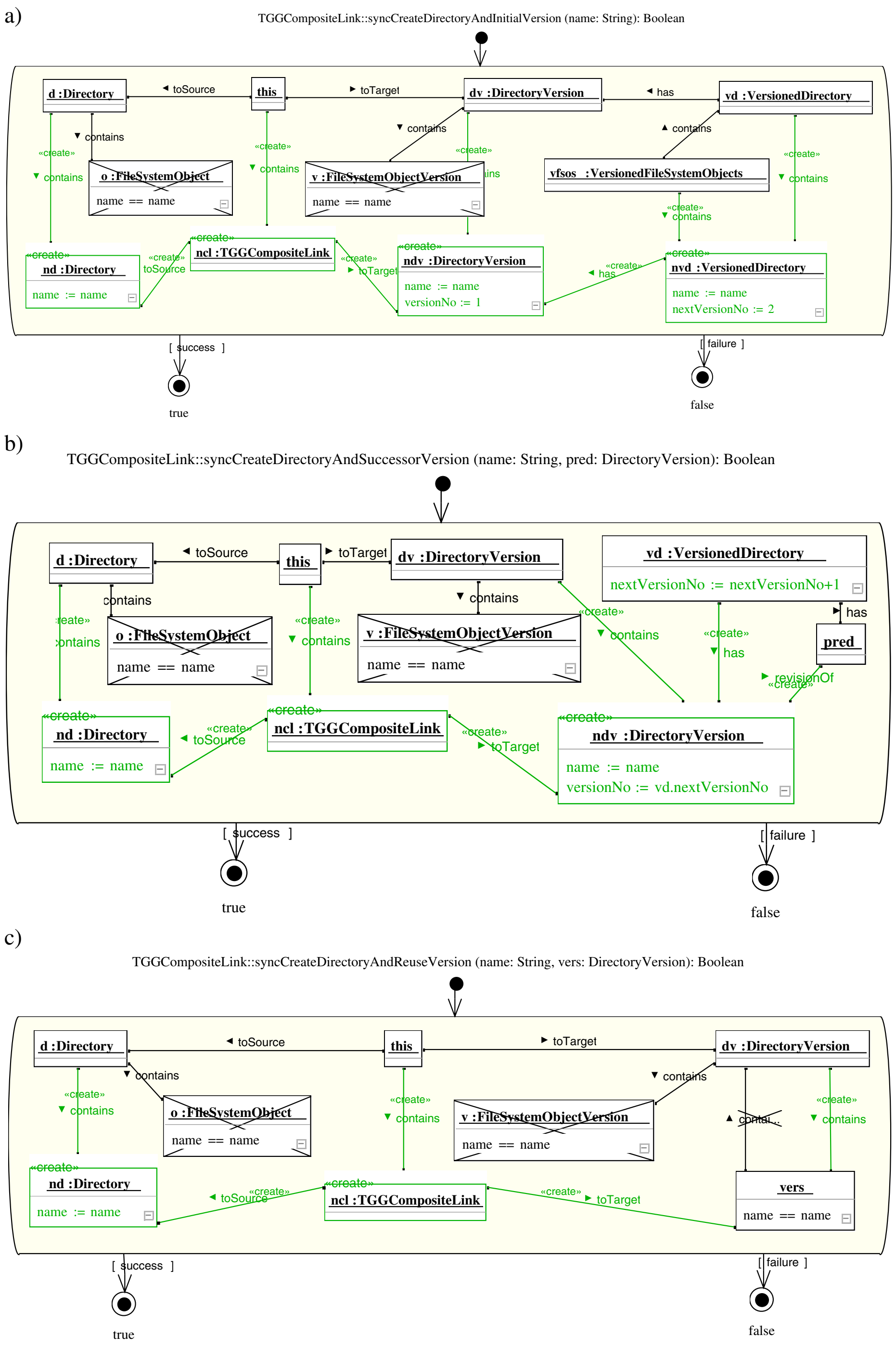


Fig. 6. Synchronous rules for creating directories and directory versions

Since all rules shown in Figure 6 are identical with respect to the source graph and the correspondence graph, the generated rules stand in *conflict*: A new subdirectory can be transformed by any of these rules, and applying one rule invalidates the other choices. Since the rules have different effects, the generated TGTS is *non-deterministic*, resulting in an integration tool which requires user interaction. In contrast, the commands introduced in Section 2 operate in a *deterministic* way. For a new file system object, the user may not deliberately choose either to create a new versioned object and its root version, or to create a successor version, or to reuse an already existing version. Rather, only the first option is available for a new file system object. A successor version is created when the file system object has already been linked to a version in the repository and has been changed locally in the workspace. Finally, a version is reused when a new version of the parent directory has been created, the child object is already under version control and has not been changed in the workspace. Similar problems occur with respect to backward rules; these cannot be elaborated here due to space restrictions.

Another problem which we encountered in our case study consists in the assumption of the TGG approach that forward and backward transformations operate *symmetrically*: From a single synchronous rule, symmetric forward and backward rules are derived. However, forward and backward transformations behave differently in our case study. For example, when the content of a file is modified in the workspace, a new version is created in the repository. In contrast, when the new version is propagated into another workspace by running an update command, the file in the workspace is overridden. Furthermore, composition hierarchies are treated differently in the workspace (tree) and in the repository (acyclic graph). As a consequence, an acyclic graph is *unfolded* into a tree when it is exported into a workspace, and a tree in the workspace is *folded* into an acyclic graph when changes are committed into the repository.

Consistency Checks and Repair Actions. One of the most important goals of the TGG approach is to relieve the modeler from the burden of specifying modifications and deletions. Since the TGG rules define only graph extensions, this abstraction works only when all operations concerning modifications and deletions can be derived automatically. To this end, generic support for *consistency checks* and *repair actions* is required (see Figure 1). Unfortunately, to the best of our knowledge providing such checks and repair actions in a generic way still constitutes an open research problem.

In our case study, consistency checks are performed in the checkStatus command. Some parts of the status checks could be covered by a generic approach, e.g., differences between values of attributes such as file names and file contents. However, there are some parts which are highly domain-specific. For example, changes in some file system object need to be propagated bottom-up to the root such that new versions in the repository may be created top-down in the course of a commit. Furthermore, the status check has to recognize updates in the repository that have to be propagated into the workspace. This is a domain-specific requirement, and there is no triple rule from which we could derive such a check. Repair actions are performed in the commands for synchronization, namely update and commit. Again, these repair actions are domain-specific. For example, when a file in the workspace is deleted, its corresponding version in the repository is not deleted, as well. Rather, a new version of the parent directory is created which does not contain this file version.

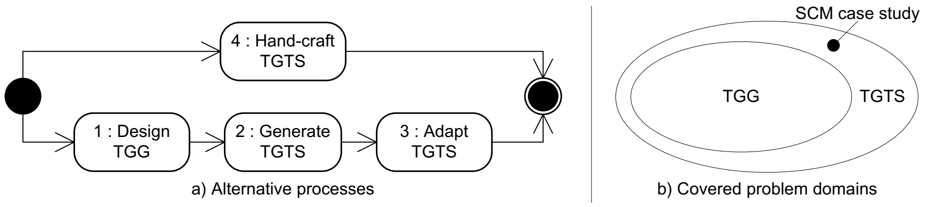


Fig. 7. Triple graph grammars or triple graph transformation systems?

5 Related Work

The overall goal of the MOD2-SCM project is to provide a model-driven product line that allows to construct a wide range of customized SCM systems with acceptable effort. These goals are related to a few research projects which were dedicated to the development of a *uniform version model*. Both ICE [9] and UVM [10] proposed rule-based generalizations of conditional compilation as a low-level, common base mechanism. To date, only a few approaches have been dedicated to *model-driven development of versioning systems* [11,12]. However, these approaches are confined to structural models inasmuch as the behavior is hard-coded into the respective system.

Triple graph grammars were introduced as early as 1994 [4]. The QVT standard [1], which was developed much later, is based on similar concepts. In the context of this paper, it is interesting to note that QVT defines both a high-level declarative and a more low-level operational language. Several projects were built upon the concepts of TGGs, but actually developed a hand-crafted TGTS [13,14]. Frameworks supporting code generation for TGGs have emerged only recently [15,16,17]. Surveys of the current state-of-the-art in TGGs are given in [5,18]. In [19], research challenges and open problems are discussed. Four design principles for TGGs are postulated: completeness, consistency, efficiency, and expressiveness. The case study presented in this paper primarily challenges the expressiveness of TGGs (see the conceptual problems reported in Section 4).

6 Conclusion

For the synchronization between repositories and workspaces, we discussed the alternatives of hand-crafting a TGTS or generating it from a TGG (Figure 7a). Following the TGG process of Figure 1, the costs of step 2 would be zero (automatic step) and step 3 would be obsolete. However, this process did not work in our case study; it was more effective to hand-craft the TGTS (step 4) than to define a TGG and adapt the generated TGTS. In fact, the case study belongs to the range of problems which are not suited for applying the TGG approach (Figure 7b). For various reasons, the operational behavior of generated rules does not match the requirements of our case study. Thus, the region $TGTS \setminus TGG$ is not empty. Improvements from theory may reduce this region further, but more practical case studies are also needed to explore the potentials and limitations of the TGG approach.

References

1. Object Management Group Needham, Massachusetts: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Final adopted specification ptc/07-07-07 edn. (July 2007)
2. Rozenberg, G. (ed.): Handbook on Graph Grammars and Computing by Graph Transformation: Foundations, vol. 1. World Scientific, Singapore (1997)
3. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.): Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools, vol. 2. World Scientific, Singapore (1999)
4. Schürr, A.: Specification of graph translators with triple graph grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 361–375. Springer, Heidelberg (1995)
5. Königs, A., Schürr, A.: Tool Integration with Triple Graph Grammars - A Survey. Electronic Notes in Theoretical Computer Science 148, 113–150 (2006)
6. Buchmann, T., Dotor, A., Westfechtel, B.: MOD2-SCM: Experiences with co-evolving models when designing a modular SCM system. In: Proceedings of the 1st International Workshop on Model Co-Evolution and Consistency Management, Toulouse, France (2008)
7. Zündorf, A.: Rigorous object oriented software development. Technical report, University of Paderborn, Germany (2001)
8. Conradi, R., Westfechtel, B.: Version models for software configuration management. ACM Computing Surveys 30(2), 232–282 (1998)
9. Zeller, A., Snelting, G.: Unified versioning through feature logic. ACM Transactions on Software Engineering and Methodology 6(4), 397–440 (1997)
10. Westfechtel, B., Munch, B.P., Conradi, R.: A layered architecture for uniform version management. IEEE Transactions on Software Engineering 27(12), 1111–1133 (2001)
11. Whitehead, E.J., Ge, G., Pan, K.: Automatic generation of hypertext system repositories: a model driven approach. In: 15th ACM Conference on Hypertext and Hypermedia, pp. 205–214. ACM Press, New York (2004)
12. Kovŝe, J.: Model-Driven Development of Versioning Systems. PhD thesis, University of Kaiserslautern, Kaiserslautern, Germany (August 2005)
13. Jahnke, J., Zündorf, A.: Applying graph transformations to database re-engineering. In: [3], pp. 267–286
14. Cremer, K., Marburger, A., Westfechtel, B.: Graph-based tools for re-engineering. Journal of Software Maintenance and Evolution: Research and Practice 14(4), 257–292 (2002)
15. Becker, S.M., Herold, S., Lohmann, S., Westfechtel, B.: A graph-based algorithm for consistency maintenance in incremental and interactive integration tools. Software and Systems Modeling 6(3), 287–315 (2007)
16. Giese, H., Wagner, R.: Incremental model synchronization with triple graph grammars. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 543–557. Springer, Heidelberg (2006)
17. Amelunxen, C., Klar, F., Königs, A., Rötschke, T., Schürr, A.: Metamodel-based tool integration with MOFLON. In: 30th International Conference on Software Engineering, pp. 807–810. ACM Press, New York (2008)
18. Kindler, E., Wagner, R.: Triple graph grammars: Concepts, extensions, implementations, and application scenarios. Technical Report tr-ri-07-284, University of Paderborn, Paderborn, Germany (June 2007)
19. Schürr, A., Klar, F.: 15 Years of Triple Graph Grammars — Research Challenges, New Contributions, Open Problems. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 411–425. Springer, Heidelberg (2008)

Model-Driven Web Engineering (MDWE 2008)

Geert-Jan Houben[1], Nora Koch[2], and Antonio Vallecillo[3]

[1] Delft University of Technology, The Netherlands
[2] Ludwig-Maximilians-Universität München and Cirquent GmbH, Germany
[3] Universidad de Málaga, Spain
g.j.p.m.houben@tudelft.nl, kochn@pst.ifi.lmu.de, av@lcc.uma.es

Abstract. The MDWE 2008 workshop offered a forum to exchange experiences and ideas related to model-driven languages and systems, applied to the Web Engineering field. Presentations and discussions focused on Model Driven Architecture (MDA) for the development of web systems; the use of metamodels, UML profiles, model-to-model and model-to-code transformations for generating web applications; and the use of tools and frameworks for supporting model-driven web development.

1 Workshop Rationale and Aims

Web Engineering is a specific domain in which Model-Driven Software Development (MDSD) can be successfully applied. Existing model-based web engineering approaches already provide excellent methods and tools for the design and development of most kinds of web applications. They address different concerns using separate models (navigation, presentation, workflows, etc.) and come with model compilers that produce the application's web pages and logic based on these models. However, most of these Web Engineering proposals do not fully exploit all the potential benefits of MDSD, such as complete platform independence, metamodeling, and model transformations.

The MDA initiative introduced a new approach for organizing the design of an application into different models so portability, interoperability, and reusability can be obtained through architectural separation of concerns. MDA covers a wide spectrum of topics and issues (MOF-based metamodels, UML profiles, model transformations, modeling languages and tools, etc.). At the same time, we see a trend towards application interoperability and Web 2.0 technologies and richer applications. However, the effective integration of all these new techniques with the already existing model-based Web Engineering approaches is still a challenge.

The fourth edition of the MDWE workshop was held in Toulouse, France, in conjunction with the MoDELS 2008 conference. The goal of the workshop was to facilitate the discussion of key issues, innovative approaches, open problems and trends in these research areas, with the aim of identifying methodologies and technologies to effectively support Model-Driven Web Engineering. The MDWE 2008 web site [1] contains all the detailed information about the workshop, including the agenda and the Proceedings with the presented papers.

2 Workshop Overview

Eight papers were selected for presentation at the workshop, from 15 initial submissions. Selected papers were published in the workshop Proceedings, which are available online [2]. The selection was based on a strict peer-review process by which each submitted paper was reviewed by at least three reviewers.

In the paper by Ernst Oberortner, Martin Vasko and Schahram Dustdar a static model is introduced, which enables the assignment of Role-Based Access Control to the pageflow at design time, achieving the integration of security concerns.

Juan Manuel González Calleros, Adrian Stanciulescu, Jean Vanderdonckt, Jean-Pierre Delacre and Marco Winckler performed in their paper a comparative analysis of model-transformation engines (publicly available, commercial, and developed ad-hoc) for the model-driven development of User Interfaces.

The paper by Ali Fatolahi, Stéphane S. Somé, and Timothy C. Lethbridge considers the use case model as a baseline to generate other models (including state machines and user interface models), which are eventually transformed into a platform-specific model used for code generation.

The work of Valeria de Castro, Juan Manuel Vara Mesa, Elisa Herrmann and Esperanza Marcos focuses on the alignment problems of models at the computational and platform specific levels (CIM and PIM), i.e. the business view in the former and the information system view in the latter.

Model-driven performance of service configurations with reliable messaging is discussed in the paper from László Gönczy, Zsolt Déri, and Dániel Varró. Starting from high-level UML models of service configurations captured by a UML profile for SOA, performance models are derived by automated model transformations in order to assess the performance cost of fault tolerance techniques.

Howard Foster, Sebastian Uchitel, Jeff Kramer and Jeff Magee also focused on services, presenting a model-driven approach for service brokering specifications.

Marco Brambilla, Piero Fraternali and Massimo Tisi introduced a transformation framework for the migration of WebML models to MDA.

The final paper by Lutzen Luinenburg, Slinger Jansen, Jurriaan Souer, Inge van de Weerd and Sjaak Brinkkemper focused on the design of a web content management systems using the method association approach.

A final session was devoted to analyse and further discuss the main issues raised in the workshop, including MDWE for "new" web applications, a classification of systems and features, patterns, criteria for a quality model, traceability, evolution support for web applications and evolution of methodologies. Extended versions of two of the papers (the one by Bambrilla et al., and the one by Gönczy et al.) were selected for inclusion in this MoDELS 2008 Workshop Proceedings.

References

[1] MDWE 2008 web site (last visited 17.11.2008) (2008), `http://mdwe2008.pst.ifi.lmu.de/`

[2] Koch, N., Houben, G.-J., Vallecillo, A.: Proc. of the 4th Int. Workshop on Model-Driven Web Engineering (MDWE 2008) (last visited 17.11.2008) (2008), `http://CEUR-WS.org/Vol-389/`

Model Transformations for Performability Analysis of Service Configurations[*]

László Gönczy, Zsolt Déri, and Dániel Varró

Budapest University of Technology and Economics
Department of Measurement and Information Systems
H-1117 Budapest, Magyar tudósok körútja 2
{gonczy,varro}@mit.bme.hu, zsolt.deri@gmail.com

Abstract. As more and more business-critical systems are based upon services deployed over flexible and dynamic platforms like Service-Oriented Architecture (SOA), there is an increasing need for such services to meet their non-functional requirements like reliability, availability, security, etc. To achieve such objectives, these services need to be designed carefully making critical design decisions early in the development process on an architectural level.

In the current paper, we aim at carrying out a *performability analysis* of service configurations to estimate the cost of using reliable messaging techniques for services with respect to performance. Such an analysis is enabled by automated *model transformations* carried out within the VIATRA2 framework.

Keywords: Model-driven Analysis, Model Transformations, Performability Analysis, Service-Oriented Architecture.

1 Introduction

As more and more business-critical systems are based upon services deployed over flexible and dynamic platforms like Service-Oriented Architecture (SOA), there is an increasing need for such services to meet their non-functional requirements like reliability, availability, security, etc. To achieve such objectives, these services need to be designed carefully making critical design decisions early in the development process on an architectural level.

Non-functional requirements of services are frequently captured by service-level agreements (SLA), which capture the required service parameters between two business parties. Unfortunately, there is currently no design time guarantee that these SLAs will actually be met during service operation. In fact, properly setting up service configurations is a complex task as performance and reliability requirements can be contradicting: an inappropriate setup of reliability attributes may cause significant decrease in performance.

[*] This work was partially supported by the SENSORIA European project (IST-3-016004).

In the current paper, we aim at carrying out a *performability analysis* of service configurations to estimate the cost of using reliable messaging techniques for services with respect to performance. Such an analysis is enabled by automated model transformations carried out within the VIATRA2 framework. Previous work [11] is extended with the technicalities of model transformations used for performability analysis. Our approach for analyzing service configurations is in line with the model-driven service engineering framework proposed by the SENSORIA research project [20].

Modeling of service configurations. This paper uses UML models conforming to the UML4SOA profile (developed within SENSORIA) as a modeling notation. UML4SOA uses standard UML extensibility mechanism and the structural part of this modular profile is closely related to the core SOA metamodel presented in [2]. The profile models several other aspects (policies, dynamic behaviour, etc.), here we rely on the non-functional extension for service design. UML4SOA defines a general description format for (design time and runtime) non-functional parameters of services. In this paper, we specialize this for reliable messaging standards; the metamodel used here is based upon the work in [12].

Performability analysis. From such service configuration models, automated model transformations generate *formal process models* for the PEPA framework (Performance Evaluation Process Algebra, [7]) to provide an early performability evaluation and prediction for service configurations with reliable messaging (Sec. 3).

In contrast to performance analysis methods which investigate the behaviour of composed services (e.g., business workflows), here we derive performance models from the SLA specification of service configurations by using basic building blocks for modeling services communicating over a reliable messaging middleware. Performability model is automatically derived from high-level system description by means of model transformations.

Model analysis transformations. These transformations were implemented in the VIATRA2 framework [22] by following a model-driven approach using parametrized transformations to derive target analysis models. A brief insight to the actual transformations is provided in Sec. 4.

2 Modeling SOA with Reliable Messaging

In the current section, we present how service configurations can be modeled using a high-level UML model dedicated to service design by a corresponding UML profile, which was designed as part of the SENSORIA project. This profile is conceptual follow up of [2] where a semi-formal platform-independent and a SOA-specific metamodel (ontology) was developed to capture service architectures on various levels of abstraction in a model-driven development process for business-level services. The UML4SOA profile includes means to capture non-functional aspects of services on a high-level of abstraction (i.e. independently of specific non-functional parameters such as availability or performance). In this section, we briefly overview the core ideas behind this modeling language.

Moreover, we specialize this general-purpose non-functional profile to capture service configurations with reliable messaging based upon a recently developed metamodel [12].

2.1 Running Example

In this paper we will use the "*On Road Assistance*" scenario developed in scope of the Automotive Case Study [15] within the SENSORIA [20] project, which describes a car-to-infrastructure application scenario.

In this scenario, the built-in diagnostic system of a car reports a severe failure of the engine which triggers the in-vehicle diagnostic system to perform an analysis of the sensor values. If the car is no longer drivable the system sends a message with the diagnostic data and the GPS data of the vehicle to the car manufacturer or service center. Based on availability and the driver's preferences, the service discovery system identifies and selects the appropriate services in the area: repair shop (garage), tow truck and rental car. The selection of services takes into account personalized policies and preferences of the driver. Upon confirmation, the owner of the car has to deposit a security payment before being able to order services.

This scenario raises several non-functional requirements against the system, as collected in [8]. In this paper, we concentrate on the *accountability*, which means on the service architecture level that the effect of communication faults have to be eliminated by the underlying middleware based upon appropriate service configurations to guarantee the message delivery between components.

2.2 A Core SOA Metamodel and Non-functional Extensions

The UML4SOA profile [16] was developed in the SENSORIA project to capture the abstract structural, behavioral and non-functional aspects of service-oriented applications. The profile is built in a modular way, and thus here, we mainly focus

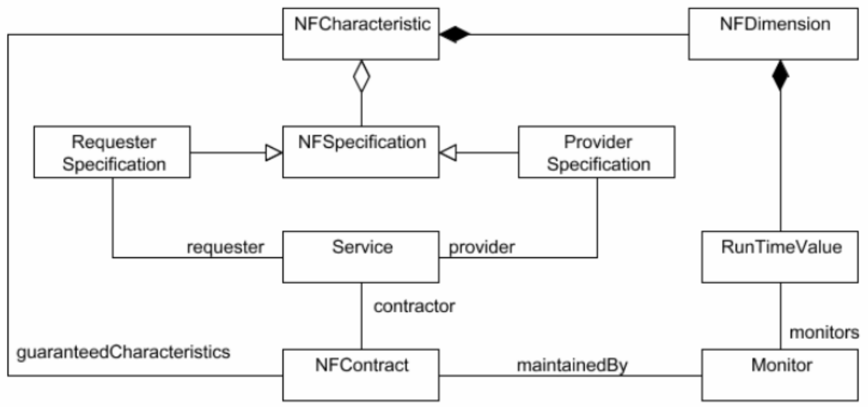


Fig. 1. Metamodel of non-functional properties of services

on non-functional aspects, which are most relevant for the current paper (see Fig. 1). These non-functional aspects were inspired by standard UML extensions (such as [17]). On a very abstract level, we define Services which are provided by Components. Each service defines a provided interface and a required interface. Each service defines a Protocol while each component has an Implementation.

Non-functional aspects are included in the UML4SOA profile by generalizing Service Level Agreements. Attributes of a service are described by NFDimensions which are collected to NFCharacteristics, which represent logical groups of properties, such as security, performance or reliable communication. These characteristics are contained within an NFSpecification.

During the operation of services, provided and requested properties of services are negotiated (which process is out of the scope of the current paper). After the negotiation, a contract with the agreed specification is created. Fulfillment of the contract is monitored by a dedicated component.

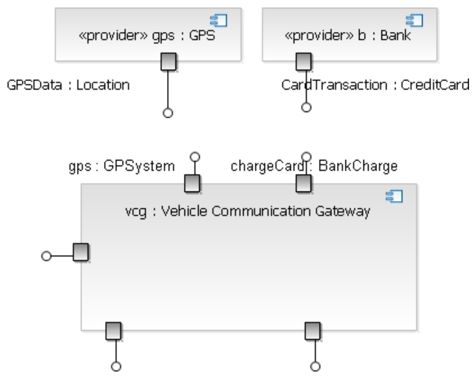


Fig. 2. Core components of the OnRoadAssistance scenario

Case study. An extract of the components of the "*On Road Assistance*" scenario [15] is shown in Fig. 2. In the current paper, we focus on the Vehicle Communication Gateway component, which is responsible for the car-to-infrastructure communication, i.e. it manages communication between external service providers, like the Bank and the global positioning system (GPS). This way, the Vehicle Communication Gateway acts as a communication mediator between a central service Orchestrator component and the actual external services. For our initial investigations, we disregard from this Orchestrator, and focus only on the other three components.

2.3 Reliable Messaging Standards for Web Services

There are various industrial standards reflecting the emerging need for reliable Web services middleware from which we focus on reliable messaging standards (e.g., WS-Reliability and WS-ReliableMessaging) in this paper.

Reliable messaging in the fields traditional distributed systems is closely related to the guaranteed semantics of message delivery. As a demonstrative example, we discuss that *at least once delivery* semantics. In the case of normal operation, every message is transferred at least once, with the possibility of sending multiple instances of the same message. This can only be allowed in systems where this does not have an undesired side-effect.

The following attributes are required for the configuration of reliable messaging (besides **messagingSemantics**, which selects the messaging mode as described earlier):

– **inactivityTimeout:** (integer, seconds), after this period of time if no acknowledgment message has arrived, the connection is closed;
– **exponentialBackoff:** (boolean), if it is set to true, time amounts between retransmissions are following an exponential distribution;
– **acknowledgementInterval:** (integer, seconds), amount of time elapsed before sending acknowledgement message;
– **retransmissionInterval:** (integer, seconds), after this time a request is resent by client if no acknowledgement arrived.

We incorporate these attributes to the UML4SOA profile by prescribing that the NFCharacteristic of an NFSpecification should contain an NFDimension specific to reliable messaging ReliableMessaging (when reliable messaging is required by a contract).

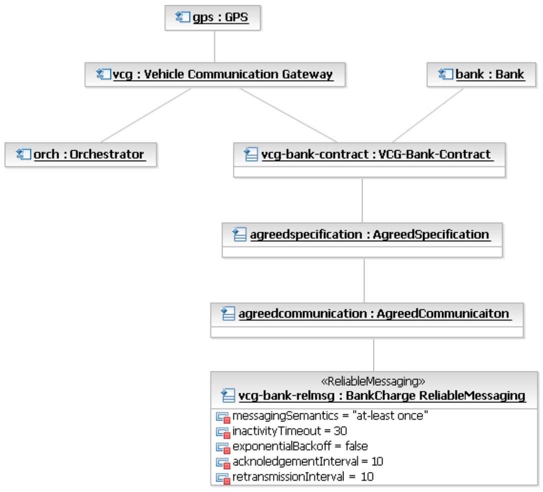

gps : GPS
vcg : Vehicle Communication Gateway
bank : Bank
orch : Orchestrator
vcg-bank-contract : VCG-Bank-Contract
agreedspecification : AgreedSpecification
agreedcommunication : AgreedCommunicaiton
«ReliableMessaging»
vcg-bank-relmsg : BankCharge ReliableMessaging
messagingSemantics = "at-least once"
inactivityTimeout = 30
exponentialBackoff = false
acknoledgementInterval = 10
retransmissionInterval = 10

Fig. 3. Non-functional specification of the communication

Case study. Reliable messaging specifications using non-functional extensions of the UML4SOA profile are captured in Fig. 3 (for charging the bank account of the driver) which prescribes that communicating with the bank requires reliable message communication.

NF Specifications are defined to describe the relevant properties of communication between Bank Charge service and Vehicle Communication Gateway etc. These specification can contain different characteristics like availability, performance or reliable messaging. Certain parameters can be defined for non-functional attributes (such as average ResponseTime, messageSemantics etc.) in course of modeling which can be used for example for generation of configuration files as can be seen later.

In the current paper, we illustrate our approach by using the *at-least-once* reliable messaging semantics as a communication model. However, using other reliable messaging semantics would not cause significant complications for the presented approach.

3 Model-Based Performability Analysis of Services

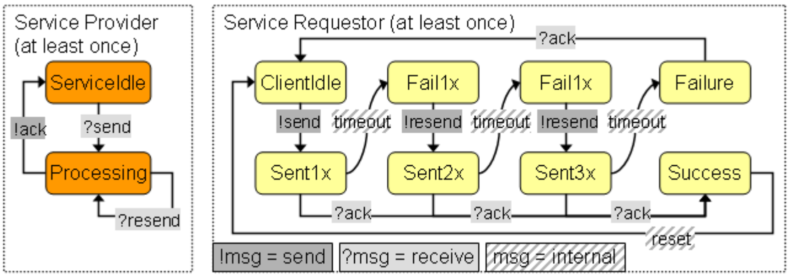
We present a model-driven technique for the performability analysis of service configurations with reliable messaging. *Performability* refers to the behavior of the system in the presence of faults, in other words, the cost of fault-handling (or fault-tolerant) techniques in terms of response time. For our investigations, we use the PEPA (Performance Evaluation Process Algebra) toolkit [7], which offers a formal language for capturing and powerful stochastic analysis techniques for the evaluation of performance models.

3.1 The Performability Model

For capturing the performability model of the basic components (in our case, the client and the server), we use a visualized version of the process algebra notation of PEPA. Each process is visualized as an automaton. Rectangles represent states, while transitions between states correspond to communication actions. ! stands for sending a message and ? means receiving a message. Sending and receiving messages is carried out by synchronization between the two processes. Internal actions without communication between processes are also distinguished (e.g., timeout trigger events). The firing frequency of transition in PEPA are considered to follow an exponential distribution.

Fig. 4 shows the stochastic performability model created as a combination of some core processes. The model represents the behavior of a service provider (Bank) and a service requester Vehicle Communication Gateway when reliable messaging is required between them.

The *service provider* (shortly, server) component is either processing a request or waiting for a new one, with the action of sending an acknowledgement to the client once the message was successfully received.



```
// Rates
rateSend = 3.0;
rateResend = 8.0
rateAck = 10.0;
rateTimeout = 1.0;
rateReset = 6.0;
//Definitions
ServiceIdle = (send, infty).Processing;
Processing = (ack, rateAck).ServiceIdle +
             (resend, infty).Processing;
```

```
ClientIdle = (send, rateSend).MsgSent1;        // send msg. ->transition to Sent1
MsgSent1 = (ack, infty).Success + (timeout, rateTimeout).Failed1; // Success or Failed
Failed1 = (resend, rateResend).MsgSent2;       // rateResend - exp. distribution par.
MsgSent2 = (ack, infty).Success + (timeout, rateTimeout).Failed2;
Failed2 = (resend, rateResend).MsgSent3;
MsgSent3 = (ack, infty).Success + (timeout, rateTimeout).Failure;
Success = (reset, rateReset).ClientIdle;
Failure = (ack, infty).ClientIdle;
ClientIdle[5] <send,resend,ack> ServiceIdle[3]  // synch. on send,resend,ack msg.
```

Fig. 4. PEPA process model of a service configuration with at-least-once messaging

The *service requester* (or shortly, client) is assumed to behave according to the *at-east-once* semantics (with an upper limit of three on the number of messages). The automaton of the service requester represents that after sending a message, it waits for an acknowledgement until a timeout occurs. Then it resends the request until the maximum number of retransmission is reached. This is represented by the non-trivial multiplication of a basic automaton as many times as the maximum number of allowed retransmissions. If an acknowledgement arrives, the message transmission is considered successful.

3.2 Performability Analysis Objectives

The typical questions for the PEPA solvers investigate *passage time* (i.e., the expected response time of the system as a function of stochastic parameters), *utilization* of states (what percentage of total operating time is spent in a particular state). In addition, *sensitivity analysis* can also be performed to estimate the effect of changing transition rates on system-level performability attributes. The number of possible retransmissions is also an interesting parameter to investigate, however, this needs the modification of the structure of the performability model (by re-executing the transformation), while tuning of other parameters requires to modify only the rates in the generated PEPA model. Core examples for using PEPA are available in [7,24].

We can investigate the utilization of states in order to answer questions like *"What percentage of time is spent waiting for the answer of the request?"*. With the parameter settings of our running example (described in Fig. 4), PEPA derives that in a steady state, the system spends 23% of the time with fault handling within states MsgSentX and FailX, which are exactly the states required for providing reliable messaging.

Sensitivity Analysis. Fig. 5 shows the (relative) change of failure rate as a function of RAMP related parameters acknowledgement time and retransmission interval based upon PEPA calculation. For the failure rate, utilization of Failure

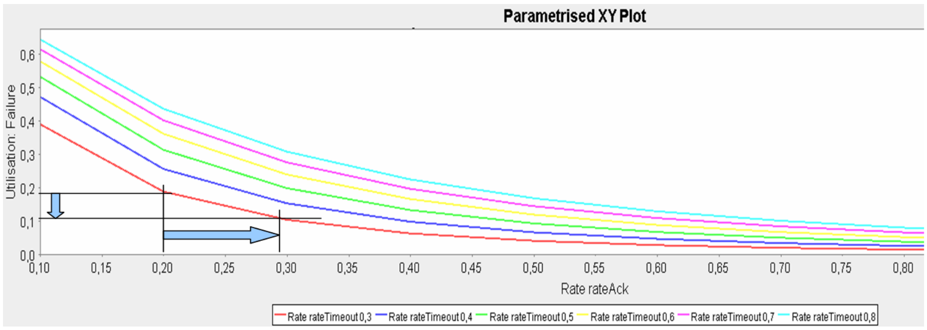


Fig. 5. Effect of RAMP-related parameters on failure

state has been used. X-axis shows different values of acknowledgment time while the different curves plot different timeout thresholds.

Our analysis results can be interpreted as early prediction of performability. For instance, one can deduce from Fig. 5 that if the rateAck rate is increased from 0.2 to 0.3 (namely acknowledgement interval decreases), then there is about 100% decrease in the frequency of errors. So it is worth improving performance of the provider if its cost is linear. Decreasing rateTimeout rate (curves with different colors) also leads to the improvement of failure rate.

We also extend our performability model to handle service configurations where a service provider itself needs to call some other (third-party) service in order to serve its own request. This intermediate (mediator) component acts as a server and a client at the same time, thus we derive its behavior by combining the basic elements of our performability model (exemplified in Fig. 4) by synchronizing the core automata on send and ack messages.

4 Model Transformation Development

Transformation overview. Essentially, automatic model transformations derive PEPA processes from the UML models of service configurations extended with reliable messaging attributes. This transformation takes various inputs:

- *Service configuration models*, which only contain the architectural design, i.e. the dependencies between services being relevant for performability analysis. For performability analysis, we identify the main roles of service providers and requesters potentially chained to incorporate third party services.
- *Predefined library of component behavior*, which captures the core, performability related behavior of reliable messaging for each party (e.g. service provider, requester). This library should include technology-related behavior, which is typically observable but not controllable (in case of a reliable messaging middleware, the overhead of access to stored message content before generating new message instances).
- *Reliable messaging parameters*, which affect both the structure and the dynamic behavior of performability models. This includes quantitative characteristics of faults of message transmission (encoded implicitly into rates of transitions) to estimate the effect of unreliable communication layer.

Transformation chain. The translation of the UML model to PEPA code was implemented in multiple steps as shown in Fig. 6 using model-to-model and model-to-text transformations implemented in the VIATRA2 framework. The input of the transformation chain is a standard UML model using UML4SOA for modeling services and non-functional parameters of messaging captured in an EMF representation. This model is first imported to the internal representation of the VIATRA2 tool.

Then uml2pepa is executed to transform relevant parts of the service model (from the performance aspect) to the concepts of the PEPA tool by taking the contracts attached to the model and generating PEPA automaton. This

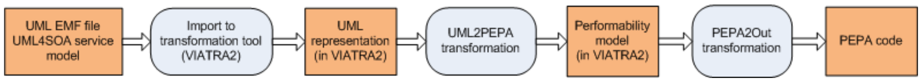

UML EMF file UML4SOA service model
Import to transformation tool (VIATRA2)
UML representation (in VIATRA2)
UML2PEPA transformation
Performability model (in VIATRA2)
PEPA2Out transformation
PEPA code

Fig. 6. Transformation chain

transformation also uses a 'parameter library' (currently encoded as a set of constants) which represent typical settings of the reliable middleware. These are also used to set default values for parameters which were uninitialized in the high level model.

Finally, pepa2out is a syntactical transformation which generates textual code from the PEPA model. Separating the syntax generation from the semantical mapping enables to develop transformations which are easier to maintain; moreover, the abstract performance model can also serve as the basis of creating input to other stochastic analysis tools.

The transformation chain has also been integrated to the SENSORIA Development Environment (SDE). SDE is an Eclipse-based tool which integrates model-driven tools for SOA development, analysis and deployment in a "SOA-style" [19] where all models and tools can be managed within the same Eclipse environment, and different tools are invoked as services.

4.1 Definition of Model Elements: Models and Metamodels

During the implementation of model based analysis technique, we first need to define the structure of the data we want to handle at different levels of abstraction, e.g., at the level of engineering models, level of formal representation and level of concrete analysis tool syntax. The performability analysis transformations we discuss are part of a complex transformation suite helping the model-based development of services with non-functional requirements.

The following code shows parts of the metamodel of the PEPA language implemented in VIATRA. We use the concept of *Entity*, *Relation* and *subtypeOf* to build hierarchical models.

```
entity(metamodel.PEPA) {
    entity(meta) {
        entity(Component);
        entity(State);
        entity(InitialState);
        subtypeOf(InitialState, State);
        relation(part, Component, State);
      // creating operators (choice, cooperation) ...
      // creating actions ...
      //creating parameters (rates) ..
      }
}
```

Besides the metamodel of the source (UML) and the target (PEPA) languages, a *reference model* also needs to be created in order to "track" the connections between corresponding source and target elements of model. Using reference models improves the maintainability of the transformation. In this case, the reference model stores information about the ”semantical” connections among UML component diagram elements and PEPA elements.

4.2 Definition of Basic Mappings: Graph Transformation Rules

Model transformations are also captured in a textual way by using a combination of (i) graph patterns for querying models, (ii) graph transformation rules for elementary model manipulations, and (iii) abstract state machines for assembling complex transformations from simple rules.

Graph patterns are the atomic units of model transformations. They represent conditions (or constraints) that have to be fulfilled by a part of the model space in order to execute some manipulation steps on the model. A *negative application condition* (NAC) prescribes contextual conditions for the original pattern which are forbidden in order to find a successful match.

Graph transformation (GT) [6] provides a high-level rule and pattern-based manipulation language to implement basic mappings. In VIATRA2, graph transformation rules may be specified by using a *precondition* (or left-hand side – LHS) pattern determining the applicability of the rule, and a *postcondition* pattern (or right-hand side – RHS) which declaratively specifies the result model after rule application. Elements that are present only in (the image of) the LHS are deleted, elements that are present only in the RHS are created, and other model elements remain unchanged. Further actions can be initiated by calling any ASM instructions within the *action* part of a GT rule, e.g. to report debug information or to generate code.

```
// pattern for finding components with
// reliable messaging specification
gtrule component2componentR(in Component, out Interface) = {
 precondition pattern lhs (Component, Interface) = {
 structure.Port(Port) in Component;
 structure.Interface(Interface) in Component;
 structure.PortType(PortType) in Component;
 structure.Port.isInstanceOf(Instance, Port, PortType);
 structure.PortType.provides(P, PortType, Interface);
 RelMsgSpecification(RelMsg);
 RelMsgSpecification.providerSpec(PSpec, RelMsg, Port);
}
action {
let PepaComponent = undef in
new (PEPA.Component(PepaComponent) in PEPA.instances);
}
```

This transformation fragment shows a graph transformation rule matching for a component and a corresponding interface which have a specification on the reliability of the messages (and therefore is a subject of the analysis).

4.3 Assembling Complex Transformations

Graph transformation is a declarative mechanism to define relations among elements in different modeling languages. However, a real model transformation also frequently necessitates efficient control structure to assemble complex model transformation programs from elementary transformation rules. VIATRA2 uses Abstract State Machines for this purpose [3]. The following code snippet describes the high-level transformation flow, which first applies transformation rule componentPattern to create basic automaton for the server-side process (which waits for incoming requests, as illustrated on Fig. 4), and then its server-side and client-side subautomata are populated.

```
machine uml2pepa {
 rule main(in Model) =
  let Model=ref(Model), PepaComponent=undef in
  forall Component, Interface below Model with apply
  component2componentR(Component, Interface, PepaComponent)
  do seq {
    // creating server process (in separate unit)
    call createServerProcess(PepaComponent);
    // creating client process (in separate unit)
    call createClientProcess(PepaComponent, ...);
  }
}
```

4.4 Platform Specific Data Library

The transformation relies on the knowledge of messaging semantics (specification level) and the messaging standards defining operations of the concrete middleware. In order to incorporate performability concepts, we need to create a platform specific library of state machines with timing parameters, which is then the basis of analysis model generation. In our case, this library is used to create the structure of performance models on the basis of reliable messaging specifications. For instance, the *at-least-once* messaging semantics with the specified value of 3 for *maxNumberOfRetransmission* will correspond to the automata in Fig. 4.

In this case, the success and failure states of the client automaton are created first, then the intermediate states are created in a recursive manner.

```
// VIATRA rule for creating the client-side automata
rule createClientProcess (in PepaComponent , in Retransmisson)
= let SIdle = undef, Success = undef, Failure = undef in
 seq {
  new (PEPA.meta.InitialState(SIdle ) in PepaComponent);
  new (PEPA.meta.State(Success) in PepaComponent);
  new (PEPA.meta.State(Failure) in PepaComponent);
  call createSentStates(PepaComponent,Retransmisson);
}
```

```
// rule for intermediate states
rule createSentStates(in PepaComponent, in Retransmisson =
let Sent=undef, Fail=undef in seq {
 if (Retransmisson > 0) seq {
  new(PEPA.meta.State(Sent) in PepaComponent);
    new(PEPA.meta.State(Fail) in PepaComponent);
    call createSentStates(PepaComponent, Retransmisson-1);
    }
}
```

The execution of the transformations on the case study model resulted in PEPA models which were then analyzed as described in Sec. 3.

5 Related Work

A framework for automated WSDL generation from UML models is described in [21], using the UML extensions of MIDAS [4]. In [13], Web service descriptions are mapped to UML models, and (after using visual modeling techniques) a composite service can be created for which the descriptor is automatically generated. However, none of these works considers non-functional properties of Web services.

Non-functional aspects of e-business applications are discussed among others in [1], having some description of deployment optimization for J2EE applications, but without discussing details of model-based deployment.

Integration of non-functional aspects in the development by model transformations is also investigated in [5,18] and [14], focusing on parts of the engineering process. However, none of these approaches address performability analysis in the context of SOA.

In a service-oriented environment, PEPA has already been used to analyze (application-specific) high-level UML models or workflow-like descriptions of services with Service Level Agreement attributes [24]. In this approach, the authors investigate performance parameters (compared to performability in our case). However, the main essential difference is the (performance-related) behavior of services needs to be modeled explicitly on the UML-level. In contrast, our technique relies only on architectural level UML models, and the core building blocks of (business-independent) performability-related behavior are instantiated in accordance with the UML model of service configurations, which allows better reusability.

Concerning previous work of the same authors, verification of the behavior of the system (i.e., checking the conformance to requirements on reliable messaging) was performed in [12], thus giving a formal semantics which can be checked by using some basic verification techniques and tools. The process model used for performability analysis in Sec. 3 is derived as an abstraction of this work, concentrating on quantitative measures. A high-level initial overview of the current framework were introduced in [9]; however, the current paper contains significantly more details, and the performability analysis is completely novel. [10] shares some conceptually similar ideas in order to carry out a model-based performance evaluation to analyze BPEL processes with SLA requirements by using DEEM.

6 Conclusions

In the paper, we presented an integrated model-driven framework for the design and analysis of standards-compliant service configurations supporting reliable messaging.

We provided an overview of transformation development in the VIATRA2 framework [23]. On the example of this transformation, we illustrated the development of metamodels and transformations using the combination of graph pattern matching mechanism and Abstract State Machine rules.

As modeling front-end, the IBM Rational Software Architect v7 UML tool was used with appropriate model importers for VIATRA. We used an SLA-like description and (in contrary to existing methods were the original source model had to be enriched by performability parameters) we created a quantitative model which is the basis of precise analysis, helping the designer to estimate the cost and benefit of using reliable middleware. Our analysis uses the PEPA toolkit.

The same model representation is used as the basis of deployment transformations to standard-compliant platforms [9], such as IBM RAMP, Apache Axis2 extended with Sandesha module and SCA environments with implementations of the Policy framework.

A thorough scalability analysis of our approach on a real-life example is also part of our research activities. From the model transformation aspect, we also plan to work on the back-annotation of the results to the engineering model.

References

1. Balogh, A., Varró, D., Pataricza, A.: Model-based optimization of enterprise application and service deployment. In: Malek, M., Nett, E., Suri, N. (eds.) ISAS 2005. LNCS, vol. 3694, pp. 84–98. Springer, Heidelberg (2005)
2. Baresi, L., Heckel, R., Thöne, S., Varró, D.: Style-based modeling and refinement of service-oriented architectures. SoSyM 5(2), 187–207 (2006)
3. Börger, E., Stärk, R.: Abstract State Machines. A method for High-Level System Design and Analysis. Springer, Heidelberg (2003)
4. Caceres, P., Marcos, E., Vera, B.: A MDA-based approach for web information system development. In: Workshop in Software Model Engineering (WiSME@UML 2003) (2003)
5. Cortellessa, V., Marco, A.D., Inverardi, P.: Software performance model-driven architecture. In: SAC 2006: Proceedings of the 2006 ACM symposium on Applied computing, pp. 1218–1223. ACM Press, New York (2006)
6. Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.): Handbook on Graph Grammars and Computing by Graph Transformation. Applications, Languages and Tools, vol. 2. World Scientific, Singapore (1999)
7. Gilmore, S., Tribastone, M.: Evaluating the Scalability of a Web Service-Based Distributed e-Learning and Course Management System. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 214–226. Springer, Heidelberg (2006)
8. Gnesi, S., ter Beek, M., Baumeister, H., Hoelzl, M., Moiso, C., Koch, N., Zobel, A., Alessandrini, M.: D8.0: Case studies scenario description, SENSORIA Deliverables Month 12 (2006)
9. Gönczy, L., Ávéd, J., Varró, D.: Model-based deployment of web services to standards-compliant middleware. In: Pedro Isaias, I.J.M., Nunes, M.B. (eds.) Proc. of WWW/Internet 2006 (ICWI 2006). Iadis Press (2006)
10. Gönczy, L., Chiaradonna, S., Di Giandomenico, F., Pataricza, A., Bondavalli, A., Bartha, T.: Dependability evaluation of web service-based processes. In: Horváth, A., Telek, M. (eds.) EPEW 2006. LNCS, vol. 4054, pp. 166–180. Springer, Heidelberg (2006)

11. Gönczy, L., Déri, Z., Varró, D.: Model-Based Performability Analysis of Service Configurations with Reliable Messaging. In: Koch, N., et al (eds.) Proc. Model Driven Web Engineering (MDWE). CEUR, vol. 389 (2008)
12. Gönczy, L., Kovács, M., Varró, D.: Modeling and verification of reliable messaging by graph transformation systems. In: Proc. of the Workshop on Graph Transformation for Verification and Concurrency (ICGT 2006). Elsevier, Amsterdam (2006)
13. Gronmo, R., Skogan, D., Solheim, I., Oldevik, J.: Model-driven web services development. In: Proc. of the IEEE Int. Conf. on e-Technology, e-Commerce and e-Servie (EEE 2004), pp. 42–45. IEEE, Los Alamitos (2004)
14. Jonkers, H., Iacob, M.-E., Lankhorst, M.M., Strating, P.: Integration and analysis of functional and non-functional aspects in model-driven e-service development. In: EDOC, pp. 229–238 (2005)
15. Koch, N., Berndl, D.: D8.2.a: Requirements Modelling and Analysis of Selected Scenarios - Automotive Case Study, SENSORIA Deliverables Month 24 (2007)
16. Koch, N., Mayer, P., Heckel, R., Gönczy, L., Montangero, C.: D1.4.a: UML for Service-Oriented Systems, SENSORIA Deliverables Month 24 (2007)
17. Object Management Group. UML Profile for QoS and Fault Tolerance (2006), `http://www.omg.org`
18. Röttger, S., Zschaler, S.: Model-driven development for non-functional properties: Refinement through model transformation. In: Baar, T., Strohmeier, A., Moreira, A., Mellor, S.J. (eds.) UML 2004. LNCS, vol. 3273, pp. 275–289. Springer, Heidelberg (2004)
19. SENSORIA Development Environment home page (2007), `http://svn.pst.ifi.lmu.de/trac/sct`
20. SENSORIA FP6 IST project (2005), `http://sensoria-ist.eu`
21. Vara, J.M., de Castro, V., Marcos, E.: WSDL Automatic Generation from UML Models in a MDA Framework. In: NWESP 2005, p. 319. IEEE, Los Alamitos (2005)
22. Varró, D., Balogh, A.: The model transformation language of the VIATRA2 framework. Science of Computer Programming 68(3), 214–234 (2007)
23. VIATRA2 Framework at Eclipse GMT, `http://www.eclipse.org/gmt/`
24. Wirsing, M., Clark, A., Gilmore, S., Hölzl, M., Knapp, A., Koch, N., Schroeder, A.: Semantic-Based Development of Service-Oriented Systems. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 24–45. Springer, Heidelberg (2006)

A Transformation Framework to Bridge Domain Specific Languages to MDA

Marco Brambilla, Piero Fraternali, and Massimo Tisi

Politecnico di Milano, Dipartimento di Elettronica e Informazione
P.za L. Da Vinci, 32. I-20133 Milano - Italy
{marco.brambilla,piero.fraternali,massimo.tisi}@polimi.it

Abstract. The Model Driven Architecture aims at the integration of different modeling languages, artifacts and tools in a unified technical space. Pre-existing development methodologies based on Domain Specific Languages (DSL) require a complex process to benefit from this integration. After a MOF metamodel for the DSL is defined, there is no standard methodology to move legacy models and tools to the new architecture. This paper proposes a general model-driven integration procedure for pre-MDA DSLs. The procedure, given the definition of suitable model transformations, is completely automatic. The proposed framework is fully implemented, in a way independent of the specific DSL that must be transformed. As a case study, a toolsuite based on WebML, a DSL for designing Web applications, is bridged to MDA.

1 Introduction

Model Driven Architectures (MDA) are becoming the platform of choice for the design and implementation of a variety of software systems. Many tools are flourishing in the opensource community, within academic research projects, and on the commercial market, fostering the wider adoption of MDA.

On the other hand, these trends put at risk of obsolescence the approaches based on pre-existing domain specific languages (DSL) and models (DSM) that do not fully comply with the MDA guidelines. These legacy development methodologies would require a complex transformation process to fully benefit from MDA. The transformation process must be designed and implemented almost from scratch for every pre-MDA design approach, with a high associated cost. This may lead to dropping the DSM instead of integrating it in the MDA framework, thus loosing modeling experiences and domain expertise.

In this paper, we discuss a general MDA framework to move DSMs specified in a legacy technical space to a MDA-compliant architecture. We propose a multi-step transformation approach based on Higher Order Transformations (HOT) to transform the legacy DSMs to MDA models whose metamodel can be either automatically or semi-automatically derived. The contribution is to offer a migration path from a legacy technical space to MDA, based on a set of transformations that are independent of the specific DSL. These transformations can be reused to:

1. transform any DSL defined in the legacy technical space to a corresponding MOF metamodel (metamodel transformation, M2T);
2. map every model defined by means of the legacy DSL to a MDA model (model transformation, M1T);
3. guarantee that the generated model is an instance of the corresponding metamodel. This can be achieved by enforcing the coherence between the two previous tasks. To this purpose, we use a Higher Order Transformation (HOT) for automatically generating the M1T transformation from the M2T transformation.

Once the framework is in place, the developer can optionally implement a final refinement transformation (M1Tr) to address particular issues of the specific DSL or specific structures that require ad hoc domain knowledge and cannot be automatically inferred.

Thanks to our approach, only one higher order transformation and one metamodel transformation are needed for each technical space. The model transformations for any DSL in that technical space can be automatically generated. Moreover, any change in the DSL does not require one to modify the transformations, because the manually written ones (M2T and HOT) depend only on the technical space, while the model transformation (M1T), the only one that depends on the DSL, can be automatically regenerated.

Besides the general framework, the paper also presents a concrete implementation in the Eclipse Modeling Framework, that provides a mature implementation for some key MDA standards. The framework is developed using the ATL transformation language, focusing on the migration from a XML/DTD technical space to Ecore-based artifacts. The implementation is used to port WebML [9] legacy models to MDA.

The paper is organized as follows: Section 2 describes the generic transformation approach, Section 3 presents the concrete transformation case, Section 4 discusses the related works and Section 5 concludes the paper.

2 Transformation Framework

The migration of the syntactical structure of a DSM from a legacy technical space to MDA is a task that involves three related issues, illustrated in Figure 1 as mappings between artifacts in the two spaces:

1. the DSMs are generally expressed in a DSL that does not conform to MOF, thus requiring the derivation of an ad-hoc MOF metamodel;
2. the DSMs have to be transformed into models in the MDA technical space;
3. the *conforms to* relationship that connects models to their language must be mapped from the legacy technical space to the MDA technical space, to ensure that the two *conforms to* relationships have the same semantics.

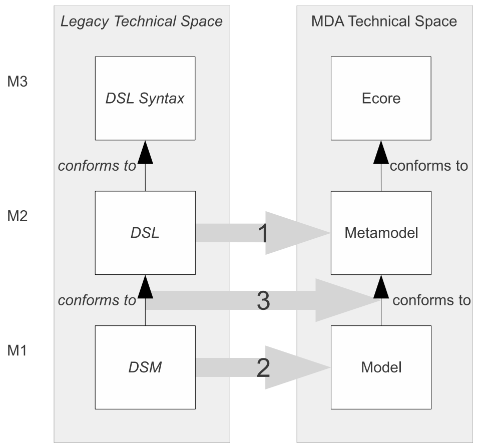


Fig. 1. Correspondences between technical spaces

The correspondences of Figure 1 are not independent. Normally, mapping (1) and (2) are designed independently, in such a way to implicitly satisfy mapping (3). As a consequence, any change in one of the mappings impacts the others and requires their adjustment. For example, a change in the mapping between the DSL and its corresponding MOF metamodel (1), preserves the *conforms to* relationship (3), but impacts the mapping between models (2). Maintaining coherence manually is a time-consuming and error-prone activity. The framework proposed in this paper allows the automatic synchronization of the transformations. In particular, we formally define two of these correspondences, namely (1) and (3), and automatically generate the other, i.e. (2), by means of HOTs.

As shown in Figure 2, the organization of the framework consists of three phases, followed by a final optional step:

1. The *metamodel generation* phase addresses mapping (1), by performing the automatic translation of the DSL to a MDA metamodel. The translation involves a first step of injection and a second step of metamodel transformation (M2T). These steps require the availability of the metametamodel (MMM) of the involved technical spaces, i.e. the legacy metametamodel and Ecore. The legacy MMM needs to be expressed as an MDA metamodel (i.e., conforming to Ecore). The DSL injector parses the concrete syntax in which the DSL is expressed and derives a representation of the DSL abstract syntax as an instance of the legacy MMM. Subsequently the transformation M2T is defined as a set of transformation rules that map the syntactical constructs of the legacy MMM to Ecore. The application of the M2T transformation translates any metamodel in the legacy technical space into a correspondent Ecore metamodel. Notice that the M2T transformation relies only on a mapping between the two MMMs, i.e. the two technical spaces. Once this

transformation has been specified, it can be reused for the migration *of any DSL* between the addressed technical spaces.
2. The *model generation* phase addresses mapping (2), by automatically translating the legacy DSMs into models compliant with the new metamodel. This phase is analogous to the previous one, but applied at a lower level in the MDA stack: it again involves an injection step followed by a transformation step. The injection step performs the parsing of the concrete syntax of the DSM, and generates an instance of the metamodel associated with the DSM syntax. Subsequently, the model transformation step (M1T) computes the final MDA model as an instance of the metamodel produced by the metamodel generation.
3. The *higher order transformation* phase addresses mapping (3), guaranteeing the coherence between the *conforms to* relationship of the two technical spaces. This task is performed in two sub-tasks: 1) a *promotion transformation* obtains the DSL metamodel by promoting the model M1 resulting from the model generation phase to metamodel (M2); 2) a manually defined HOT derives the M1T transformation by translating the M2T transformation, and eventually analyzing the structure of the DSL metamodel.
4. Finally, the *refinement model transformation* can be optionally applied to adapt the resulting model to some manually introduced variations of the DSL metamodel. This phase typically affects marginal aspects of the automatically generated models, and will not be treated in detail in the rest of the paper.

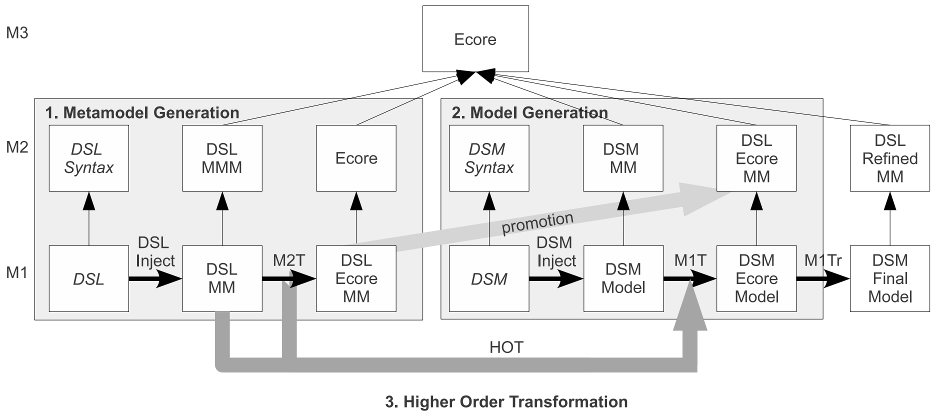


Fig. 2. Diagram of the general framework

3 Case Study: Modernization of a DTD-Based DSL

This section describes the case study on which we built our prototype framework. We introduce and discuss the DSL technical space and the transformations of the aforementioned phases of Metamodel Generation, Model Generation, and Higher Order Transformation. The framework implementation is orchestrated by means of an Ant script that uses the tasks of the AM3 project [1]. The complete sources of the prototype framework can be downloaded from [3].

3.1 The Technical Space: DTD/XML

The case study focuses on the migration from a legacy technical space based on XML files that conform to given Document Type Definitions (DTD). Figure 1 can be specialized for the use case: *(i)* the DSL syntax is defined by the DTD grammar (level M3); *(ii)* the DSL is specified by a DTD document (level M2); *(iii)* the DSM is an XML document (level M1); *(iv)* the *conforms to* relationship corresponds to the *validity* relationship in the DTD/XML technical space, i.e. to the relationship that connects a valid XML document to its associated DTD.

Moving from the DTD/XML technical space to the MDA technical space requires the following mappings:

– *M3 mapping:* to map the DTD grammar to Ecore, associating to each DTD construct (e.g. ATTLIST) a correspondent Ecore translation (e.g. a set of EAttributes).
– *M2 mapping:* to map a specific DTD document to a correspondent Ecore metamodel, associating each DTD definition (e.g. a specific ATTRIBUTE) to a correspondent Ecore element (e.g. a specific EAttribute).
– *Conformance mapping:* to map the *validity* relationship to the *conforms to* relationship, so that if an XML document is valid with respect to its associated DTD then its correspondent model conforms to its metamodel.

The difference in expressive power between the DTD syntax and Ecore makes the bridging between these formalisms a non-deterministic activity, since the DTD syntax is ambiguous in several points, compared to Ecore[19]. This means that, when mapping a DTD construct to an Ecore translation, the M3 mapping involves choices among different suitable alternatives, that can only rely on default policies and heuristics. Examples of lack of expressiveness are the general CDATA attribute type, that can be mapped on different Ecore types such as EString, EInteger, or EFloat; and the IDREF attribute type that can be mapped as an EReference without specifying the associated eType.

The discussion about the optimal policies for the M3 mapping is outside the scope of this paper. The most convenient heuristics can be different depending on the considered DSL. The framework we provide assures that, upon changes on the M3 mapping (or on the DSL itself), the M2 mapping is automatically synchronized, thus maintaining the coherence between M3 and M2.

3.2 The DSL: WebML

The sample DSL considered in our case study is WebML [9], a DSM language for data-, service-, and process- centric Web applications. It allows specifying

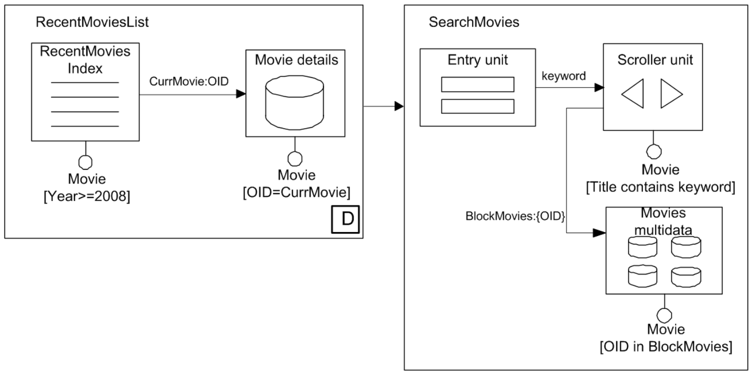


Fig. 3. Example of WebML hypertext model

the conceptual model of Web applications built on top of a data schema and composed of one or more hypertexts used to publish or manipulate data. The specification consists of several sub-models: the *data model* represents the data schema; the *hypertext model* represents the content of pages, the navigation paths, and the parameters that flow among the components; and the *presentation model* describes the visual aspects of pages.

The data model is the standard Entity-Relationship (E-R) model. Upon the same data model, different hypertext models (*site views*) can be defined (e.g., for different types of users or for different publishing devices). A site view is a graph of *pages*, consisting of connected *units*, representing data publishing components: a unit displays instances of an entity, possibly restricted by a *selector*. Units are related to each other through *links*, representing navigational paths and carrying parameters. WebML allows specifying also update *operations* on the underlying data (e.g., the creation, modification and deletion of instances of entities or relationships) or operations performing arbitrary actions (e.g. sending an e-mail, invoking a remote service, and so on).

Figure 3 shows a simple hypertext, containing two pages. Page *Recent Movies List* contains an index unit defined over the *Movie* entity, which shows the list of movies produced after year 2008 (selector [Year > 2008]) , and a data unit also defined over the *Movie* entity, which displays the details of the movie selected from the index. The link between the two units carries the parameter CurrMovie, used by the selector [OID=CurrMovie] of the data unit. Another link connects *Recent Movies List* page to *Search Movies* page, without carrying any parameter. Page *Search Movies* contains an entry unit for inserting the movie title to be searched, a scroller unit, and a multidata unit displaying a block of search results. Through the scroller unit it is possible to move to the first, previous, next, and last block of results.

The WebML language is supported by the WebRatio CASE tool [4], a development environment for the visual specification of Web applications and the automatic generation of code for the J2EE platform.

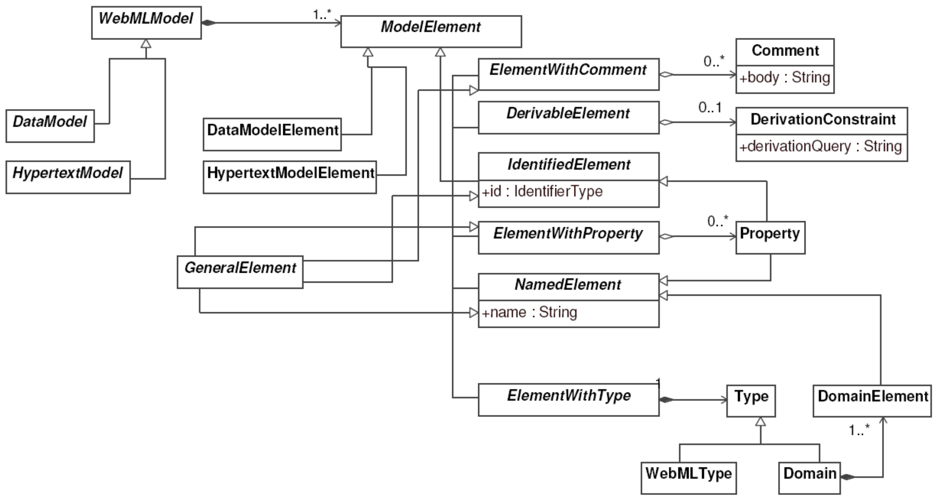


Fig. 4. Overview of the WebML metamodel

Some proposals of WebML metamodels already exist ([8] and [19]). We have extended the metamodel presented in [17] as summarized in Figure 4, which is further refined for describing all the details of the DSL.

3.3 Metamodel Generation

Figure 5 depicts the general diagram of the Metamodel Generation phase, that transforms a DSL specified by means of a DTD into an Ecore metamodel. The DTD metamodel is a refined version of the one provided in [14]. The first class objects of this metamodel are Element, Attribute, Sequence, etc.

DSL Injection. The DSL Injection step consists in parsing the concrete syntax of the DTD specification to derive an instance of the DTD metamodel. The AM3 framework provides support to custom injectors developed as a Java class implementing the org.eclipse.m2m.atl.engine.injectors.Injector interface. Developing a custom Java injector for the legacy metametamodel is generally a simple task, because a parser of the concrete syntax of the metametamodel is usually available in the legacy technical space and can easily be extended with semantic actions to build the correspondent metamodel elements. In our work, a DTDInjector class has been developed using the DTD Parser provided in [2].

M2T. The M2T transformation defines the M3 mapping, between the DTD MM and Ecore. M2T is implemented as an ATL transformation defining the translation policies between the elements of a DTD and the classes of a metamodel. M2T is defined without any domain specific knowledge: it is a general transformation that can be reused to translate every DSL defined by means of

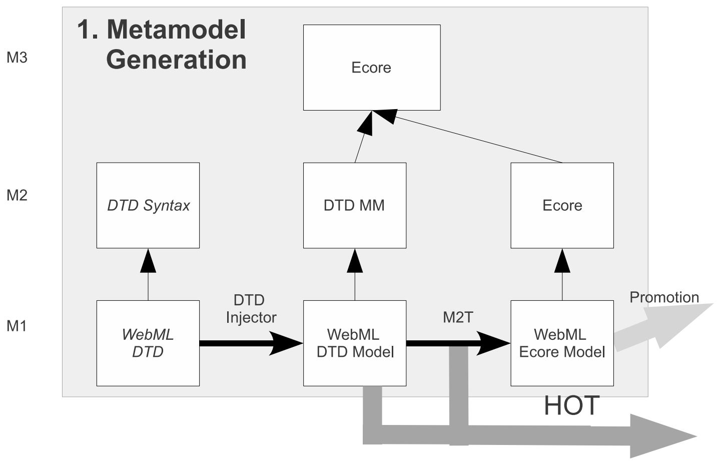


Fig. 5. Case Study: Metamodel Generation

a DTD into an Ecore Metamodel. Following is an excerpt from the M2T implementation in the case study:

```
module m2t; -- Metamodel Transformation
create OUT : MOF from IN : DTD;
rule Element {
 from
  element : DTD!RestrictedElement
 to
  class : MOF!EClass (
   name <- element.name,
   eStructuralFeatures <-
    element.content.attributes.union(element.content.children)
  )
}
rule EmptyElement {
 from
  element : DTD!EmptyElement
 to
  class : MOF!EClass (
   name <- element.name,
   eStructuralFeatures <- element.content.attributes
  )
}
rule Attribute {
 from
  attribute : DTD!Attribute
 to
  attr : MOF!EAttribute (
   name <- attribute.name
  )
}
```

The rules translate DTD Elements, Attributes and Children to EClass, EAttribute and EReference elements. This version of the M2T transformation contains several heuristic choices. For instance:the Children of a DTD Element are always translated as containment references in Ecore; and DTD Attributes are translated to simple EAttributes, without considering their type. The latter heuristic is sufficient for simple cases, when the use of IDREFs is limited and can be dealt with in the M1Tr transformation.

3.4 Model Generation

Figure 6 shows the diagram of the Model Generation phase, that transforms a WebML project specified as an XML document into an instance of the WebML metamodel generated in the Metamodel Generation phase. The core is an XML injection step, followed by the generated transformation M1T.

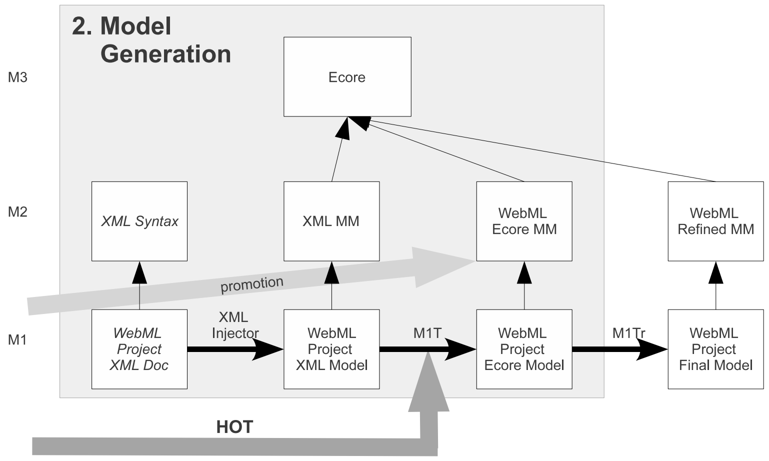


Fig. 6. Case Study: Model Generation

The metamodels involved in this phase are the XML metamodel and the WebML metamodel. The former is a standard metamodel provided by the EMF project whose first class objects are Tag, Node, Attribute, etc.

DSM Injection. The injection of the DSM is easily performed by means of the XMLInjector, a standard injector provided by the AM3 project, to convert an XML document to an instance of the XML metamodel.

M1T. M1T is the transformation that maps an XML model to an Ecore model, instance of the generated DSM metamodel, i.e. the WebML metamodel. Being an ATL transformation that has the DSL metamodel as the output metamodel, M1T can not be independent of the DSL metamodel. For this reason, traditional transformation-based approaches to the migration of DSMs to MDA require one to develop a different M1T transformation for each DSL. The generative approach that we propose overcomes this problem: in our framework M1T is still a DSL-specific transformation, but it is generated by a DSL-agnostic HOT.

An exemplary ATL rule, part of the M1T transformation, is:

```
rule INDEXUNIT {
 from
  element : XML!Tag (name = 'INDEXUNIT')
 to
  result : DSLMM!INDEXUNIT (
   [...]
  )
}
```

The example represents the skeleton of the M1T translation rule for an INDEXUNIT WebML component. In the XML representation of a WebML project, an INDEXUNIT is represented by an INDEXUNIT tag as a child of the container PAGE. The ATL excerpt matches every XML INDEXUNIT tag and generates an instance of the INDEXUNIT Class in the WebML MM (for brevity we omit the code for the structural features).

M1Tr. Being generated from M2T, the M1T transformation is a DSL-specific transformation that does not use any DSL-specific knowledge. Since M2T and HOT are DSL-agnostic transformations, any DSL-specific transformation has to be specified in a subsequent step that lies outside the generative framework, represented by the optional M1Tr (i.e. M1T refinement) transformation. M1Tr translates the generated model to an instance of a manually defined DSL metamodel that usually has only limited differences with the generated one. M1Tr is a DSL-specific transformation that can be used to solve issues such as the different expressive power of the metametamodels, the structural limitations of the DSL-agnostic transformations, and the implementation limits of the prototype (especially wrt. the M2T syntax). In the WebML case, M1Tr can adapt the generated models to the official WebML metamodel, manually designed in [17].

3.5 Higher Order Transformation

The Higher Order Transformation phase is responsible for the translation of the M2T transformation, that generates the new metamodel, to the M1T transformation, that generates the new models. The only metamodel involved in this phase is the ATL metamodel that provides a representation of an ATL transformation as an instance model.

ATL Injection/Extraction. The execution of the HOT has to be preceded and followed respectively by an injection and an extraction of the ATL transformations. M2T is injected as an instance of the ATL metamodel and, after the HOT has been executed, M1T is extracted to its textual form. The injection and extraction of the ATL transformations is implemented using the generic EBNFInjector and EBNFExtractor provided by the AM3 project. The two classes allow one to specify any concrete syntax for the injection/extraction by means of a Textual Concrete Syntax (TCS) model[15]. A TCS is an Ecore model, that conforms to the TCS metamodel, describing the syntactic rules of the concrete syntax. An ATL-TCS model is provided by the ATL project and can be used as a parameter of the generic injector and extractor.

HOT. The most original aspect of our approach is the use of a HOT to enforce the coherence of the mappings at different levels. The following code is a simplified rule extracted from our HOT:

```
module HOT;
create OUT : ATL from IN : ATL, DTD : DTDMM;
rule Classes {
 from
   matched : ATL!MatchedRule ( outPattern.elements->first().type.name = 'MOF!EClass' )
 using {
   matchedElements : Sequence(OclAny) =
     DTDMM!Element.allInstances()->
     select(e | e.oclType().toString() = 'DTDMM!'+matched.inPattern.elements->
            first().type.name ); [...] }
 to
   atl : distinct ATL!MatchedRule foreach (e in matchedElements) (
     name <- e.name,
     inPattern <- inPat,
     outPattern <- outPat,
     isRefining <- false,
     isAbstract <- false ),
   -- InPattern
   inPat : distinct ATL!InPattern foreach (e in matchedElements) (
     elements <- elementin,
     filter <- oc ),
   elementin : distinct ATL!SimpleInPatternElement foreach (e in matchedElements) (
     id <- 'tag0',
     varName <- 'tag',
     type <- intype ),
   intype : distinct ATL!OclModelElement foreach (e in matchedElements) (
     name <- 'XML!Tag' ),
   -- Filter
   oc : distinct ATL!OperatorCallExp foreach (e in matchedElements) (
     operationName <- '=',
     source <- noac,
     arguments <- s ),
   noac : distinct ATL!NavigationOrAttributeCallExp foreach (e in matchedElements) (
     name <- 'name',
     source <-fv ),
   fv : distinct ATL!VariableExp foreach (e in matchedElements) (
     name <- 'tag',
     referredVariable <- elementin ),
   s : distinct ATL!StringExp foreach (e in matchedElements) (
     stringSymbol <- e.name ),
   -- OutPattern
   outPat : distinct ATL!OutPattern foreach (e in matchedElements) (
     elements <- elementout ),
   elementout : distinct ATL!SimpleOutPatternElement foreach (e in matchedElements) (
     varName <- 'class',
     type <- outtype ),
   outtype : distinct ATL!OclModelElement foreach (e in matchedElements) (
     name <- 'DSLMM!'+e.name )
   [...]}
```

The rules of the HOT match in their source pattern the ATL rules of the input transformation. Then the HOT rules can make use of the DSL metamodel to derive useful information on the structure of the DSL.

The previous sample rule matches the rules of M2T that generate Ecore Classes in the target metamodel. When one of those rules is matched, the *using* part of the rule saves in the matchedElements variable all the instances matched by the transformed rule. Finally a set of output ATL rules is generated, by iterating on the matchedElements variable.

The small excerpt alone generates a minimal skeleton of a rule without considering any of the structural features of the output pattern. It can be easily shown that when the Classes rule is applied to the WebML metamodel (in particular to the INDEXUNIT element), it translates the M2T shown in Section 3.3 exactly in the M1T rule shown in Section 3.4.

An approach based on a high level of abstraction may lead to an increased development cost. In particular, the complexity of the HOT grows with the expressive power of the language in which M2T is specified. A first approach to face this issue is limiting the ATL features supported by the HOT to a defined set. For example, our prototype imposes some constraints on M2T:

- only matched (declarative) rules are allowed;
- the source pattern of each rule must comprise one source pattern element (as in ATL 2004);
- the target pattern of each rule must comprise one simple or iterative target pattern element;
- local variable sections or imperative block sections are not allowed;
- OCL declarative expressions are restricted to basic path expressions and to some basic operations on collections, such as union.

While these restrictions did not hamper the translation of a DTD into Ecore, more advanced constructs might be needed for complex technical spaces or heuristics. In this case, the HOT transformation can be extended or, in the worst case scenario, implemented with a general purpose language (e.g., Java).

4 Related Work

The framework described in this paper extends the preliminary work presented in [7]. The paper [7] concentrates on the DSL metamodel, with special focus on the modeling of data derivation rules, and only sketches the transformation project; this work details the general transformation framework and explains in greater detail how to use it to bridge any DSL to MDA.

The issue of defining a bridge between specific technical spaces and MDA has been addressed in several works, such as [6], [5], [16] and [22]. The only approach that uses HOTs is [6], which describes the migration from the Microsoft DSL technical space to the Eclipse Modeling Framework. Even if the problem and the chosen means are quite similar, the solution presented in [6] has remarkable differences with our proposals. In [6] the HOT is designed with the sole purpose to maintain the framework independent of the specific DSL. Our approach also aims at the independence of the DSL, but adds the requirement of automatic synchronization between the mapping of the metamodels and the mapping of the models. In practice, our solution allows one to change the mapping at the meta-level and automatically get an updated transformation at the model level. As discussed in Section 3, the extra requirement comes with a cost. Our HOT is more complex than in [6], and its development requires a limitation on the syntax of the metamodels mapping.

Analogies exist also with the co-adaptation of models compliant to an evolving metamodel ([11], [20], [13]). With respect to these works our framework is the first to use higher order transformations to directly synchronize the model level transformation with the meta level transformation.

Higher order transformations have already been used to perform various tasks in model driven development [10], [11], [12]. To our knowledge this work is the first to address a HOT that translates an ATL transformation at the meta-level to the associated transformation at model level.

Several approaches have focused on the transformation between XML and metamodels: [21] surveys 13 proposals, classified according to the direction of the transformation (i.e. forward, backward or both) and the concrete formalisms used as source/target of the transformation. To the best of our knowledge, the only approach conducting a forward transformation from DTD to MOF is [19], which focuses on the same case study, the WebML DSL. With respect to this work, our framework provides, on the basis of the transformations defined at the meta-level, the generation of coherent transformations at the model level, and thus allows the immediate reuse of existing WebML models in MDA.

Finally, some works concentrate on the specific issue of manually mapping WebML to MDA: [17] remodels WebML using MOF and [18] proposes a WebML UML 2.0 profile to facilitate the interoperability between the WebML IDEs (e.g. WebRatio [4]) and UML modelling tools. Our proposal extends these works, as it can partially automate the production of the metamodel.

5 Conclusions

In this paper we have discussed a MDA framework to move DSMs specified in a legacy technical space to a MDA-compliant architecture. The transformation framework covers the translation of both the metamodel and the model levels, and grants automatic coherence of the two. The translation specified manually does not depend on the specific DSL, but only on the technical space. In this way, we offer a flexible migration solution that is resilient to the changes of the DSL. Our implementation experience over a real DTD-based DSL called WebML showed the feasibility and the advantages of the approach. However, some drawbacks have been highlighted too. The most critical point is related to the ATL implementation of the HOT: since DTD is a fairly simple syntax definition language, the metametamodel transformation has been rather simple to implement too. Only declarative ATL constructs have been used, and no complex OCL constraints were needed. In this context, the implementation of the HOT with ATL has been a reasonable task. However, if the M2T transformation would have involved a deeper use of ATL, the implementation of the ATL HOT would have been a painful job. Indeed, if M2T is expected to be a transformation that involves the full expressive power of ATL, the HOT should be implemented with a different technique (e.g., as a Java program).

References

1. AM3 (2008), `http://www.eclipse.org/gmt/am3/`
2. DTDParser (2008), `http://www.wutka.com/dtdparser.html`
3. Framework Implementation (2008), `http://home.dei.polimi.it/mbrambil/legacytomda`

4. WebRatio (2008), http://www.webratio.com/
5. Abouzahra, A., Bézivin, J., Didonet Del Fabro, M., Jouault, F.: A practical approach to bridging domain specific languages with UML profiles. In: Best Practices for Model Driven Software Development Workshop at OOPSLA (2005)
6. Bezivin, J., Hillairet, G., Jouault, F., Kurtev, I., Piers, W.: Bridging the ms/dsl tools and the eclipse modeling framework. In: International Workshop on Software Factories at OOPSLA (2005)
7. Brambilla, M., Fraternali, P., Tisi, M.: A metamodel transformation framework for bridging webml models to mda. In: MDWE workshop in Models 2008 (2008)
8. Cattell, R.G., Barry, D.K., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T., Velez, F.: The Object Data Standard: ODMG 3.0, 1st edn. Morgan Kaufmann, San Francisco (2000)
9. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications, 1st edn. Morgan Kaufmann, San Francisco (2002)
10. Cicchetti, A., Di Ruscio, D., Pierantonio, A.: A metamodel independent approach to difference representation. Journal of Object Technology (JOT) 6(9), 165–185 (2007) (Special issue on Proc. of TOOLS Europe)
11. Fabro, M.D.D., Valduriez, P.: Semi-automatic model integration using matching transformations and weaving models. In: Proceedings of the 2007 ACM symposium on Applied Computing, pp. 963–970 (2007)
12. Graaf, B., van Deursen, A.: Using mde for generic comparison of views. In: Proceedings of 4th MoDeVVa Workshop: Model-Driven Engineering, Verification and Validation, INRIA, pp. 57–66 (2007)
13. Gruschko, B., Kolovos, D.S., Paige, R.F.: Towards synchronizing models with evolving metamodels. In: Proc. Int. Workshop on Model-Driven Software Evolution at IEEE European Conference on Software Maintenance and Reengineering (ECSMR) (2007)
14. Guyard, P.: DTD Metamodel, www.eclipse.org/gmt/am3/zoos/atlanticZoo
15. Jouault, F., Bézivin, J., Kurtev, I.: TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In: Proceedings of the 5th International Conference on Generative Programming and Component Engineering (2006)
16. Kern, H., Kuhne, S.: Model interchange between ARIS and Eclipse EMF. In: 7th Workshop on Domain-Specific Modeling at OOPSLA (2007)
17. Moreno, N., Fraternali, P., Vallecillo, A.: WebML modelling in UML. Software, IET 1, 67–80 (2007)
18. Moreno, N., Fraternali, P., Vallecillo, A.: A UML 2.0 profile for WebML modeling. In: 2nd International Workshop on Model Driven Web Engineering (MDWE) at ICWE (2006)
19. Schauerhuber, A., Wimmer, M., Kapsammer, E., Schwinger, W., Retschitzegger, W.: Bridging WebML to model-driven engineering: from document type definitions to Meta Object Facility. Software, IET 1, 81–97 (2007)
20. Wachsmuth, G.: Metamodel adaptation and model co-adaptation. In: Ernst, E. (ed.) ECOOP 2007. LNCS, vol. 4609, pp. 600–624. Springer, Heidelberg (2007)
21. Wimmer, M., Schauerhuber, A., Kapsammer, E., Kramler, G.: From document type definitions to metamodels: The WebML case study. Technical Report of Vienna University of Technology (March 2006)
22. Wimmer, M., Schauerhuber, A., Strommer, M., Schwinger, W., Kappel, G.: A semi-automatic approach for bridging DSLs with UML. In: 7th Workshop on Domain-Specific Modeling at OOPSLA (2007)

First International Modeling Security Workshop

Jon Whittle[1], Jan Jürjens[2], Bashar Nuseibeh[2], and Glen Dobson[1]

[1] Dept. of Computing, Lancaster University, Bailrigg, Lancaster LA1 4YW, United Kingdom
whittle@comp.lancs.ac.uk, g.dobson@lancs.ac.uk
[2] Computing Department, The Open University, Walton Hall, Milton Keynes, MK7 6AA, United Kingdom
j.jurjens@open.ac.uk, b.nuseibeh@open.ac.uk

Abstract. This report summarizes the results of the 1st International Workshop on Modeling Security, which was held as part of the 2008 International Conference on Model-Driven Engineering Languages and Systems (MODELS). The workshop received 21 paper submissions of which seven were chosen for presentation at the workshop. A further twelve were included in the proceedings and presented at the workshop as posters. The papers can be found by viewing the online workshop proceedings at: http://CEUR-WS.org/Vol-413/.

1 Introduction

This document reports on the 1st International Modeling Security Workshop, which was held on September 28, 2008, in Toulouse, France, as part of the 2008 International Conference on Model-Driven Engineering Languages and Systems (MODELS). The aim of the workshop was to bring together practitioners and researchers in both software and system modeling and security to transfer ideas, foster new collaborations, and define a research agenda for secure modeling of software-intensive systems.

The call for papers led to a total of 21 paper submissions, of which seven were chosen for presentation at the workshop. The workshop proceedings include an additional twelve papers, which were presented as posters. The proceedings are available online as Vol. 413 of the CEUR Workshop Proceedings: http://CEUR-WS.org/Vol-413/. The workshop website is accessible at: http://www.comp.lancs.ac.uk/modsec. The workshop was well attended with over twenty-five participants actively engaging in the two keynote talks, paper presentations and discussion sessions.

2 Workshop Theme

It is well known that ensuring computer security is one of the key challenges in making modern systems safe, reliable and dependable. The number of vulnerability exploits is rising exponentially, and estimates set the cost of security breaches at between $13 billion and $1.6 trillion per year.

Secure *programming* techniques are now generally well understood. Best practice guidelines teach programmers how to avoid buffer overflows, when to validate inputs and how to apply cryptography. Automated tools scan source code for vulnerabilities, many of which can be detected automatically.

However, large classes of attacks cannot be avoided using such methods. Insider attacks, for example, bypass authentication protocols. These sophisticated types of attacks require a more holistic view of a system's vulnerabilities and necessitate security analysis techniques that take into account all phases of development. In other words, there is a pressing need for systematic methods for analyzing and assessing the security of system models, where models here are interpreted broadly to include requirements, architecture and design, as well as organizational and business models.

This workshop brought together academics and practitioners working on the following topics related to security modeling: modeling security requirements; modeling security in design and architecture; languages for modeling security; verification and validation of security models; model-based testing for security; applications/experience of using security modeling; challenges for security modeling; and processes and methodologies which incorporate security modeling.

3 Keynote Presentations

There were two keynote presentations to open the workshop. Neil Cooke, from the University of Surrey, spoke about the Information Assurance (IA) challenge in software engineering from a National Technical Authority (NTA) perspective. Most countries have NTAs that support IA in government and set standards for products and systems used in government. The UK NTA is a significant organization of about 500 people and is based in Cheltenham.

The classical model for security is to consider functional requirements, identify the threat in terms of data value, loss impacts, mechanisms of threat, opportunity and then to identify mitigations and residual risks that have then to be managed. The move from the communications age to the information age, however, is changing the shape of the whole IA world. The problem space and functional complexity is expanding at a rate that challenges Moore's Law. It is no longer possible to evaluate solely by examining the security critical aspects because they do not exist as physical elements. Rather, they are virtualized across the whole system. In this environment, to maintain Assurance & Evaluation capability for Confidentiality, Integrity and Availability, there must be increased use of higher-level concepts, modeling and automated toolsets. Whole system modeling is fast becoming the method of choice for the evaluation and test of IA critical systems.

The second keynote was given by Ketil Stølen, Chief Scientist and Group Leader at SINTEF. Ketil spoke about the security analysis of critical infrastructures such as the electric power supply or telecommunications and noted that their security is complicated by the fact that such infrastructures are mutually dependent. He proposed a reductionistic approach to the modeling and analysis of security risk scenarios with mutual dependencies. His approach may be used to deduce the risk-level of an overall system from previous security risk analyses of its constituent systems. It may also be used to decompose the analysis of a complex system into separate parts that can be carried out independently.

4 Research Paper Presentations

The workshop included seven research paper presentations, arranged into two separate sessions.

Mukhtiar Memon presented the paper *SECTISSIMO: a platform-independent framework for security services* [1]. SECTISSIMO is a layered approach for the modeling of security-critical, service-oriented systems. Functional models are enriched with security extensions which are then transformed into executables. SECTISSIMO takes the crucial step of providing a platform that abstracts from the underlying security technology using a layer of abstract security artifacts.

Karine Peralta presented the paper *Specifying Security Aspects in UML Models* [2]. This work was performed as part of Karine's MS thesis. It concerns a technique for specifying UML security stereotypes, which aims at guiding developers in annotating vulnerable model parts and allowing automatic security test case generation.

Koen Yskout presented the paper *Transforming security audit requirements into a software architecture* [3]. This research is an approach for automated transformations from a security requirements model to a consistent architectural model. The approach can be used with an existing architectural model, and allows input from the architect to be taken into account. The transformation from audit requirements into a UML model is implemented using QVT and Eclipse EMF.

Tejeddine Mouelhi talked about *Mutating DAC and MAC Security Policies: A Generic Metamodel Based Approach* [4]. This work is a novel application of mutation testing techniques applied to the security domain. DAC and MAC security policies can be specified, implemented and then validated using mutation testing.

Adam Shostack then gave an entertaining talk based on *Experiences Threat Modeling at Microsoft* [5]. The paper describes a decade of experience on threat modeling products and services at Microsoft. Adam described the current threat modeling methodology used in the Security Development Lifecycle. This methodology is a practical approach, usable by non-experts, centered on data flow diagrams and a threat enumeration technique of "STRIDE per element". The paper covers some lessons learned which are likely to be applicable to other security analysis techniques.

The next presentation was of the paper *Using Common Criteria as Reusable Knowledge in Security Requirements Elicitation* [6], by Motoshi Saeki and Haruhiko Kaiya. The elicitation of security requirements (SRs) is a crucial issue to develop secure information systems of high quality. Although there are several methods mainly for functional requirements such as goal-oriented methods and use case modeling, most of them do not provide sufficient supports to identify threats, security objectives and security functions. This paper proposes the usage of Common Criteria and related knowledge sources to identify SRs from functional requirements through eliciting threats and security objectives.

Finally, Haralambos Mouratidis spoke about a *Curriculum for Modeling Security: Experiences and Lessons Learned* [7]. Recent research has identified that security analysis should be integrated into software engineering techniques and security should be considered from the early stages of the software systems development process. Although researchers have focused their efforts towards this direction, the educational curriculum is not properly addressing this issue. In this paper, the author presents the experiences and lessons learned from developing and running a module in Secure Software Systems Engineering at the MSc Level.

5 Concluding Remarks

This was the first workshop dedicated to the topic of Modeling Security. It sparked a great deal of interest in both the security and modeling communities. The 19 contributions showed a broad range of interesting research topics and pointed the way towards future research.

Acknowledgements

The workshop organizers would like to thank the program committee: Ruth Breu (University of Innsbruck, Austria); Neil Cooke (University of Surrey, UK); Geri Georg (Colorado State University, USA); Charles Haley (Asia Pacific University College of Technology and Innovation, Malaysia); Siv Hilde Houmb (University of Twente, The Netherlands); Wouter Joosen (Catholic University of Leuven, Belgium); Ulrich Lang (ObjectSecurity, UK); Yves Le Traon (IRISA, France); Haris Mouratidis (University of East London, UK); Utz Roedig (Lancaster University, UK); Jörg Schreck (Telefónica O2, Germany); Duminda Wijesekera, (George Mason University, USA).

References

[1] Memon, M., Hafner, M., Breu, R.: SECTISSIMO: A Platform-independent Framework for Security Services
[2] Peralta, K., Zorzo, A.: Specifying Security Aspects in UML Models
[3] Yskout, K., de Win, B., Joosen, W.: Transforming security audit requirements into a software architecture
[4] Mouelhi, T., Fleurey, F., Baudry, B., Traon, Y.L.: Mutating DAC And MAC Security Policies: A Generic Metamodel Based Approach
[5] Shostack, A.: Experiences Threat Modeling at Microsoft
[6] Saeki, M., Kaiya, H.: Using Common Criteria as Reusable Knowledge in Security Requirements Elicitation
[7] Mouratidis, H.: Curriculum for Modelling Security: Experiences and Lessons Learned

Security Requirements Elicitation Using Method Weaving and Common Criteria

Motoshi Saeki[1] and Haruhiko Kaiya[2]

[1] Dept. of Computer Science, Tokyo Institute of Technology
Ookayama 2-12-1-W8-83, Meguro-ku, Tokyo 152-8552, Japan
saeki@se.cs.titech.ac.jp
[2] Dept. of Computer Science, Shinshu University
Wakasato 4-17-1, Nagano 380-8553, Japan
kaiya@cs.shinshu-u.ac.jp

Abstract. The elicitation of security requirements (SRs) is a crucial issue to develop secure information systems of high quality. Although we have several requirements elicitation methods, most of them do not provide sufficient supports to identify security threats, security objectives and security functions. Security functions are closely related to architectural design of the information system, i.e. solution space, and knowledge from the solution space is necessary to elicit appropriate SRs of higher quality. This paper proposes the usage of Common Criteria and related knowledge sources to identify SRs from functional requirements through eliciting threats and security objectives. Our proposed technique is to weave through Common Criteria two types of elicitation methods; one is any existing functional requirements elicitation method and the other is a typical method for eliciting security functional requirements so that we can have a powerful method.

1 Introduction

Information systems deployed at different sites are being connected to each other through networks and their users can obtain various services anytime and anywhere. In this circumstance, it is very significant to protect assets in an information system from events and/or malicious actors that compromise their security and therefore it is necessary to develop the information systems with functions that protect from security threats.

In usual information system development like waterfall style, the requirements to an information system are elicited after a business process modeling stage. It is necessary to elicit the requirements to security functions (simply security requirements) as early as possible, in order to reduce the development cost and to develop the information system of higher quality [1]. Some techniques to elicit security requirements have been proposed and put into practice, e.g. misuse case [2], abuse case [3], security use case [4], the application of i* [5] and secure Tropos [6]. Almost of them are the extended versions of use case modeling and goal-oriented approaches, which are requirements elicitation ones originally for functional requirements (FRs), so that they can adapt to the elicitation of security requirements. However, since security functions are closely related to system architecture design, i.e. artifacts on a solution space of the problems, thus it is difficult to elicit appropriate security requirements without considering the system architectures. For instance, let's consider the data base system that stores university students' grades and its functions for the students to access to their grades. There is a potential of the threat that grade data of a student can be read by the others. The technique of password authentication can be adopted to mitigate the occurrences of this threat, so that the only student that is authenticated and identified can read her grade data from the data base system. Therefore, a file system of password data used for authentication and identification (password file) is newly adopted in the system and it stores pairs of student IDs and passwords. The malicious person illegally and furtively may read password data from the password file and impersonates other students to get their grade data when adopting such a technique. To mitigate this threat further, we can have a solution to encrypt the password data in the file. We can consider threats as concepts of a problem space, while the countermeasure techniques to mitigate the threats, e.g. password authentication, password file and cryptography are the concepts in a solution space of this problem domain. Thus, not only both of them are closely connected, but also a new threat (a problem) may be invented from the newly adopted solutions. This relation expresses just a twin-peak model where development activities in these two spaces [7]. Requirement elicitation activities both in a problem space and in a solution space are indispensable to appropriate security requirements elicitation. The existing studies are biased to requirements elicitation on the problem space side, and there are quite few studies that both sides are simultaneously considered. To make the methods for eliciting security requirements work well, requirements analysts should have not only knowledge of a problem space but also knowledge related to security functions on a solution space, e.g. password authentication and cryptography etc. However it may be a rare case where a requirements analyst has sufficient knowledge of both spaces, and some techniques to provide an analyst with knowledge of security functions are necessary to be weaved with an elicitation method.

In this paper, we propose an weaved security requirements elicitation method that uses some standard documents as a guideline or knowledge source while requirements elicitation on the solution space side is being performed. We select Common Criteria Part 2 [8] and its related document (ECMA-271 E-COFC) [9] as an example of knowledge source, because the Common Criteria consists of general concepts in the solution space, and it can be considered as a kind of catalog to provide knowledge on threats, security objectives and security functions that have generally appeared. For example, by using Common Criteria, we can select the objective "data encryption" from the catalog, to mitigate the threat "disclosure of password data". Furthermore, from the catalog we can select more concrete security functions, e.g. the procedures to generate encryption keys and to abandon them, in order to achieve the selected objectives. Our technique is for embedding the usage of Common Criteria to any existing functional requirements elicitation methods such as goal-oriented approaches and use case modeling ones.

The rest of the paper is organized as follows. We explain Common Criteria (CC) and describe how it can be used for security requirements elicitation in a solution space in section 2. Sections 3 and 4 show the overview of the proposed method by using the examples of goal-oriented method and use case modeling respectively, in order to show that our approach works well. The related work and future research agenda are presented in sections 5 and 6 respectively.

2 Basic Idea

2.1 Weaving Methods

Typical activities to elicit security requirements (SRs) can be considered as follows;

1. Identify the assets that should be protected,
2. Identify security threats that can attack the identified assets,
3. Identify security objectives that can mitigate the identified threats,
4. Compose security functional requirements by refining the identified security objectives.

Figure 1 shows how the above activities are embedded into a usual method of use case modeling. In the figure, the left hand side part beginning with "1. Identify actors" represents a typical activity flow of a normal use case modeling method, while the right hand side does specific activities for additionally developing misuse cases, their relationships to normal use cases such as *threaten* and *mitigate* and misuse case descriptions. As shown in the figure, the activities of these two activity flows are *weaved* and performed in order to get a use case and misuse case model as security requirements. After identifying actors and use cases, an analyst starts identifying assets to be protected. In this case, the assets to be protected are actors and/or use cases. Then the analyst gets misuse cases and mis actors as security threats that prevent the actors from performing successfully their use cases. For each misuse case, she identifies as its security objectives the use cases that can mitigate its execution, and describes the contents of the newly identified use cases.

As shown in this example, we should consider two points to be supported; 1) shifting activities between the left part and the right one in Figure 1, and 2) performing the activities in the right part. The first point shows how we can use intermediate artifacts developed in the left part to perform the right part and vise versa. We use the reusable knowledge on security functions that is included in standard documents such as Common Criteria (CC) in order to support the above two points. In the following

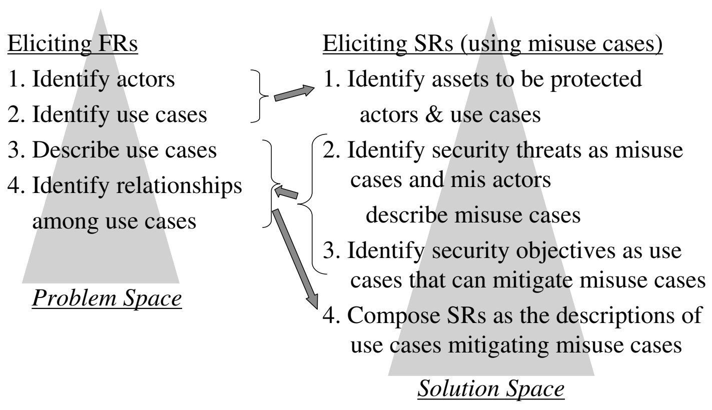

Eliciting FRs
1. Identify actors
2. Identify use cases
3. Describe use cases
4. Identify relationships among use cases
Problem Space
Eliciting SRs (using misuse cases)
1. Identify assets to be protected
actors & use cases
2. Identify security threats as misuse cases and mis actors
describe misuse cases
3. Identify security objectives as use cases that can mitigate misuse cases
4. Compose SRs as the descriptions of use cases mitigating misuse cases
Solution Space

Fig. 1. A Method of Use Case and Misuse Case Modeling

two subsections, we introduce the overview of CC and then present our approach on the usage of the knowledge included in CC.

2.2 Common Criteria

The Common Criteria (CC) is an international standard prescribing how to write the documents that are used for assessing security properties of the information system called TOE (target of evaluation). The produced document includes two types of documents; one document called *security target* is to specify security properties of the TOE, and another is to describe security assurance requirements used for verifying the compliance of the TOE product with the security properties. In this paper, we consider the former document, i.e. a security target. The CC itself consists of three parts, and we use its Part 2 as knowledge source of requirements elicitation.

A security target basically consists of 6 chapters. It begins with the overview of the TOE in chapter 1 and then describes its functional requirements in chapter 2. The description of functional requirements includes the information on the assets to be protected. In the example of the database system of students' grades mentioned in the previous section, the assets to be protected are grade data of students. Potentials of threats and assumption, etc. are described in the successive chapter 3 "Security Problem Definition". In the above example, we can consider identity spoofing by impersonation as a potential of threat. The chapter 4 "Security Objectives" provides for the objectives to mitigate the threats listed in the chapter 3. To mitigate the threat "impersonate", authentication by password can be given as a security objective. Note that each security objective described in chapter 4 should be linked to the threats in chapter 3, to clarify which objective mitigates which threats. In the chapter 5 "Security Requirements", the components of security functions to represent *exactly* the security objectives listed up in chapter 4 are described. In CC Part 2, security functional (SF) components are catalogued beforehand in the form of template. This is the reason why we use CC Part 2, a collection of SF components and their relationships, as knowledge source in our approach. We can select appropriate templates from the CC Part 2 catalog and instantiate them by filling their slots with relevant information. We represent the security objective as the combination of the instantiations of the selected templates in chapter 5. For instance, the template class FIA is for representing the security objective "identification and authentication", and it has the templates of the functional requirements descriptions relevant to "identification and authentication" such as authentication failure, user authentication, user identification, user attribute definition, etc. In chapter 6, the last chapter, we specify the security functions by logically combining and summarizing the component descriptions selected and instantiated in the chapter 5. It is necessary to maintain traceability between threats, security objectives, SF components and the specification statements of security functions.

2.3 Using Reusable Knowledge

Figure 2 shows the overview of our approach. Firstly, we mention how to use the knowledge in order to support the activities shown in the right part of Figure 2, i.e. eliciting security requirements (SRs). Although CC Part 2 has about 120 SF components as a catalog, it has no catalogs of threats and security objectives. On the other hand, ECMA-271 E-COFC [9], which can be considered as a profile of CC in a certain problem domain, includes the catalogs of threats and security objectives. In this paper, we use them together with CC Part 2. As shown in the right hand side of Figure 2, we accumulate a threat catalog, a security objective catalog and a SF component catalog, and hold relationships between their catalog entries (i.e. security objective *mitigates* threat, SF component *represents* security objective). After a threat is identified, a requirements analyst should perform the two tasks; 1) identifying the security objectives that can mitigate the threats, and 2) identifying the SF components that can represent the identified security objectives. These two tasks can be considered as those in a solution space and be supported by the catalogs. For example, suppose that the analyst selected a threat "Impersonation" for students (T.impersonate in the catalog) from the threat catalog. To mitigate it, the relationship between the threat catalog and the security objective catalog suggests that O.Authentication (authentication for students as authorized users) and O.Integrity (protection of integrity of authentication data) should be selected from the security objective catalog. Furthermore, the analyst can select SF components (templates) of FIA class (User Identification and Authentication) in order to refine and represent the security objective O.Authentication, by using the relationship *represent* between the security objective catalog and the SF component's. She fills the slots with relevant information, e.g. the acceptable time of authentication failures and the actions to be performed when the failures exceed to an acceptable value (e.g. ringing an alarm etc.) in the templates FIA_AFL family. And then she completes the document of security requirements.

Secondly, we focus on the support to weaving two activity flows, i.e. bridging between the left part and the right one in Figure 2. An artifact produced with the functional requirements (FRs) elicitation method (left part of the figure) includes the assets to be protected and they appear as elements of the artifact. For example, when we adopt use case modeling as a FRs elicitation method, the assets to be protected can appear as use cases, actors or parts of use case descriptions. To identify the possible threats to the

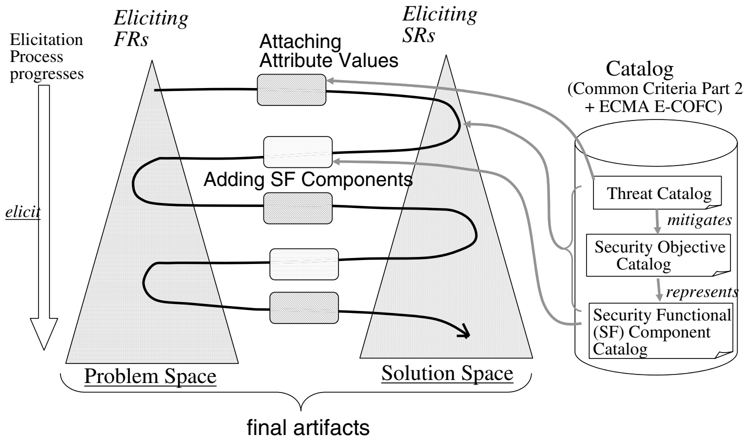

Elicitation Process progresses
elicit
Eliciting FRs
Attaching Attribute Values
Eliciting SRs
Catalog (Common Criteria Part 2 + ECMA E-COFC)
Adding SF Components
Threat Catalog
mitigates
Security Objective Catalog
represents
Security Functional (SF) Component Catalog
Problem Space
Solution Space
final artifacts

Fig. 2. Using Knowledge Included in Common Criteria

assets, we characterize the assets by attaching some attribute values to them. As shown in Figure 2, attribute values are used to identify threats. After getting SF components by performing “Eliciting SRs”, the analyst fills the slots with suitable information and add these SRs to the FRs. In the following two sections, we will clarify which attributes we use.

3 Using a Goal-Oriented Method

As the first example, we adopt a goal-oriented method as a FRs elicitation method and explain how our approach works. Figure 3 also shows an activity flow with a simple example.

Step1. Identify functional requirements (FRs) :
An analyst elicits the FRs of the information system by using the existing elicitation methods such as goal-oriented methods and use case modeling. As an example, she uses a simplified version of goal-oriented method. As shown in the figure, she decomposes the root goal “Checking grades” for the database system of students’ grades and decomposes it into two sub goals “Retrieving grades” (a student retrieves his grade data) and “Getting grades” (a student gets his grade data from the database as a result of retrieval) with AND decomposition.

Step2. Identify assets and their attributes :
The analyst identifies from the FRs the assets to be protected. In the case of using a goal-oriented method only, the assets can be extracted from goals and words appearing in the goal descriptions written in natural language. For each of the identified assets, the analyst attaches the attributes to it. Generally, we use 5W1H (Who, What, When, Where, Why and How) as the attributes to characterize the assets. The attributes to be used depend on a problem domain. In this example, the analyst focuses on the operationalized sub goal “Retrieving grades” as the asset to be protected, and its attributes are the actor *who* performs it (unspecified person), the action type of *what* the actor performs (sending retrieval information to the system) and the location *where* the actor does (a student can perform at unspecified place).

Step3. Identify threats :
The analyst infers and derives the potentials of threats from the attribute values of the assets and the threat catalog by means of inference rules prepared beforehand. In this example, since the actor of the asset (goal “Retrieving Grades”) was unspecified (persons) and the location was also unspecified, she selects T.imperonate using the catalog and an inference rule. The inference rules specify the relationships among possible threats and the attribute values of the assets. For the above example, the analyst used the following rule of “if then” style to get T.impersonate.

if who(x) = unspecified $\wedge$ where(actor(x)) = unspecified
    then select T.impersonate(actor(x)), where x is an asset.

Step4. Identify security objectives :
The analyst selects suitable security objectives from the security objective catalog, using the identified threats and the asset attribute values. In this example, she selected O.Authentication from the catalog, by applying the inference rule like:

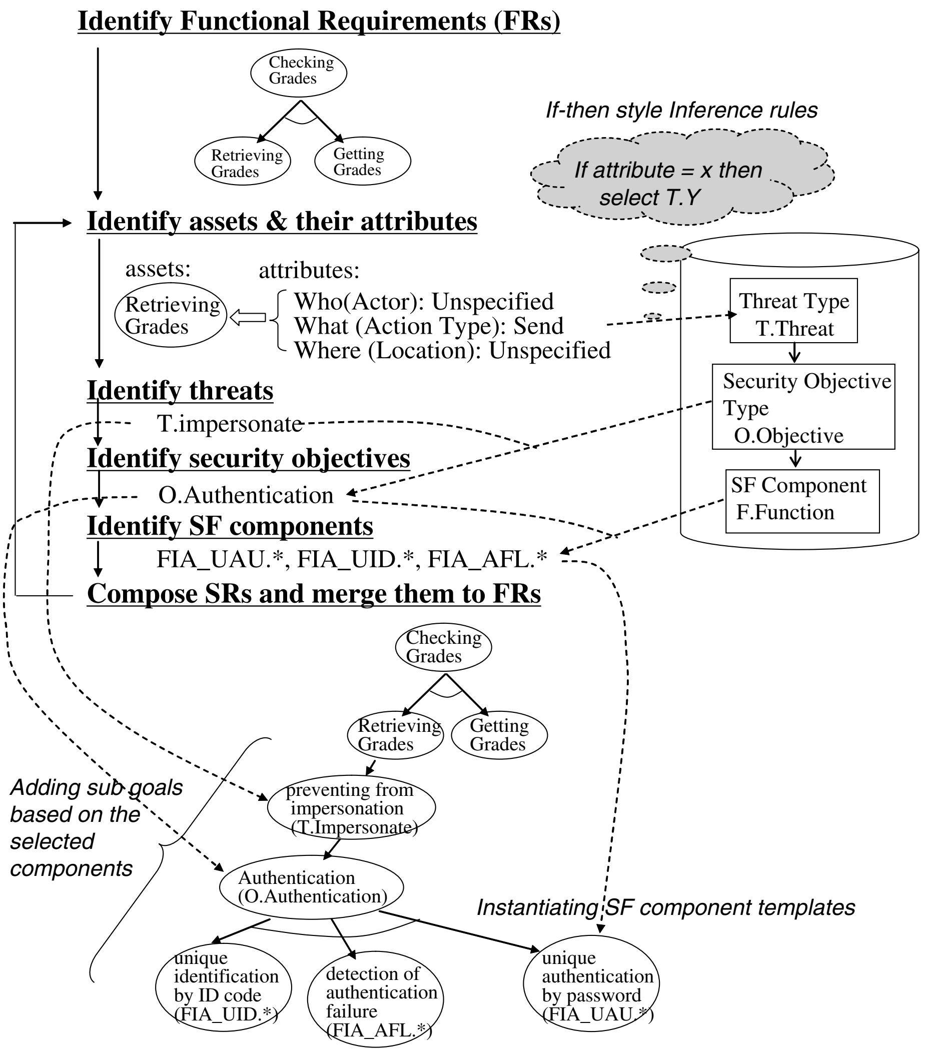

Identify Functional Requirements (FRs)
Checking Grades
Retrieving Grades
Getting Grades
If-then style Inference rules
If attribute = x then select T.Y
Identify assets & their attributes
assets:
attributes:
Retrieving Grades
Who(Actor): Unspecified
What (Action Type): Send
Where (Location): Unspecified
Threat Type T.Threat
Security Objective Type O.Objective
SF Component F.Function
Identify threats
T.impersonate
Identify security objectives
O.Authentication
Identify SF components
FIA_UAU.*, FIA_UID.*, FIA_AFL.*
Compose SRs and merge them to FRs
Checking Grades
Retrieving Grades
Getting Grades
Adding sub-goals based on the selected components
preventing from impersonation (T.Impersonate)
Authentication (O.Authentication)
Instantiating SF component templates
unique identification by ID code (FIA_UID.*)
detection of authentication failure (FIA_AFL.*)
unique authentication by password (FIA_UAU.*)

Fig. 3. Elicitation Method

if who(x) = unspecified $\wedge$ T.impersonate **then** select O.Authentication(actor(x)). This rule suggests that we could adopt the authentication technique in order to prevent some actors from retrieving the grade data of the other students by means of spoofing and impersonation.

Step5. Identify security functional (SF) components :
The analyst selects the set of the candidate SF components using the identified security objectives and the asset attribute values. This step is similar to the former steps 3 and 4, i.e. the predefined inference rules support the selection of SF components. In this example, the analyst selects FIA_UAU, which is a family of the

functional components for user authentication. Since these components have the dependencies to FIA_UID (the functions for user identification) and to FIA_AFL (the functions for measures to authentication failures), some of them are selected to refine the security objective appropriately. Dependency relationships among SF components are very helpful to avoid missing security requirements.

Step6. Compose security requirements (SRs) and merge them into FRs :
The analyst combines the identified SF components logically and makes the document of SRs from them. This document explains for the customers how the described functions can mitigate the threats and achieve the security objectives. In this example, since the analyst uses the goal-oriented approach, she adds these identified elements (threats, security objectives and the instantiated SF components) as the sub goals of the asset to be protected, i.e. "Retrieving Grades". As shown in the bottom of Figure 3, the sub goals corresponding to T.Impersonate, O.Authentication and FIA_UAU are successively added to "Retrieving Grades". FIA_UID and FIA_AFL are also added with AND decomposition in the same level because of their dependencies to FIA_UAU. To maintain traceability, these elements as sub goals and their relationships are kept in the goal graph.

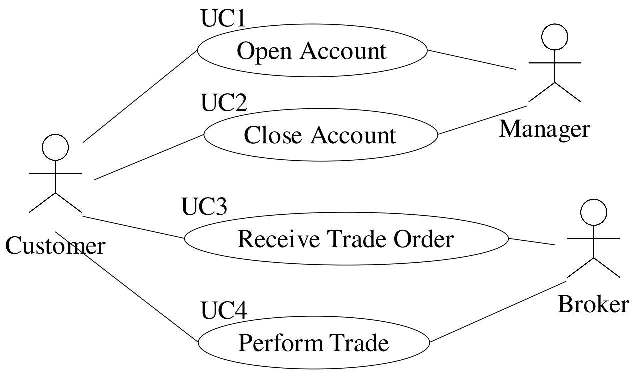
For the newly added security requirements, the analyst iterates from step 2 to step 6 and elicits new requirements. In the example of the figure, after eliciting a sub goal "unique authentication by password" from FIA_UAU, she identifies passwords as the asset to be protected at the iterated step 2, then identifies the threat T.Data_Theft (password data may be stolen) in the same way, and continues her elicitation tasks.

4 Using a Use Case Modeling Method

We illustrate another example where use case modeling is applied in order to show our approach can work on various existing requirements elicitation methods, not only goal-oriented approaches. The example in this section is the system of a financial company that provides investment services to its customers, which was included in [10]. Figure 4 shows the overall of the system with a use case diagram and a part of the use case descriptions.

This system includes many potentials of threats and some of them were pointed out in [10]. In this example, out of them, we pick up and pay attention to the threat "the manager can create a spurious authorization to access the account". More concretely, since the manager can know the identification & authentication information (ID and Password) of the customer when the customer opens his account (with UC1), a malicious manager can be spuriously authorized using the customer's ID and password, and can trade orders by impersonating as the customer. We apply our approach to this security problem.

Step1. Identify functional requirements
As shown in Figure 4, we elicit the functions of the system with use case modeling. The security problem results from the case where the customer can ask the manager to open his account, not by himself. The action 3 of the UC1 suggests that the manager could obtain the ID and the password of the customer and then impersonate the customer using UC3 with the obtained ID and password.



UC1 : Open Account
1. Manager gets the personal information from Customer
   e.g. name, telephone number, etc.
2. Manager inputs the personal information of Customer to System.
3. System issues Customer ID and Password to Manager.
4. Manager informs Customer ID and Password to Customer
5. ... (to be continued)

UC3 : Receive Trade Order
1. Customer inputs ID and password to System.
2. System authorizes right of Trade Order to Customer.
3. Customer specifies commodities and their amounts which he wants to trade.
4. ... (to be continued)

Fig. 4. Use Case Diagram and Descriptions

Step2. Identify assets and their attributes

Assets can be elicited from constructs of use case diagrams, e.g. actors and use cases, and the words in use case descriptions. We identify the asset to be protected from the use case diagram as UC3 “Receive Trade Order”, and find its attributes as follows.

who (who performs): Customer, Broker
what (what is performed): Input, Authorize, Specify, ...
how (how it is performed) : Identify and Authenticate by Customerís ID and password
where: (where it is performed) : unspecified
when : (when it is performed) : after opening an account
why: (why it is performed) : Customer gets trades

Step3. Identify threats

In this example, we use a word-matching technique instead of rigorous “if-then” inference rules used in the previous example of section 3. The attribute values are set based on the words appearing in the use case description of UC3, e.g. authorize, input, specify, etc., and we match these words with the words appearing in the catalogs. More concretely, we look for the entries whose explanation sentences include the synonyms as the attributes, i.e. Input, Authorizes, etc. Concentrating on the word "Authorize" and its flections (we use a wild card to express the word and its inflection, like authoriz*) of the "what" attribute, we can obtain the following threats from the threat catalog of ECMA-271 (E-COFC) because the explanations of all of them have the word authoriz*. T.Insider, T.Outsider, T.Secret_Disclose.

Step4. Identify security objectives

In the similar way, using the relationship "mitigates" between the catalog entries, we can get OE.Access_Malicious from the security objective catalog. In addition, we can get the followings by using the "what" attribute authoriz* and "how" attribute values identify* and authentic*.

O.Authen_Address, O.Authen_Age, O.Authen_Indep, O.Authen_Protect.

Step5. Identify security functional components

Since OE.Access_Malicious is an environmental security objective and it means that the malicious activities of the manager cause the threat, the SF components included in CC cannot mitigate it. However, we can select the FIA_UAU.3.2 to mitigate the other security objectives by using the word matching with authentic* and the relationship "represents".

Step6. Compose security requirements and merge them into FRs

The use case corresponding to FIA_UAU 3.2 is newly added as a sub use case of UC3.

Note that we used the inference rules different from the goal-oriented approach example of section 3. The reason is that use case descriptions are written in natural language and we consider that the word-match technique would be simpler and more suitable. It suggests that the rule styles, in addition to the assets, vary on the adopted FRs elicitation method.

5 Related Work

Although CC is used to assess whether the security properties of the IT products reach a certain standard level or not, it can provide reusable knowledge for security requirements elicitation such as SF component catalog. Our technique focuses just on this point. The approach proposed by Ware et. al. was the first one to use CC as a reusable catalog to support the elicitation of security requirements [11]. In their approach, an analyst constructs a profile for each actor after drawing a use case diagram and correlates the actors to threats based on the actors' profiles only. Although our approach can be considered as its more elaborated version, it can deal with wider requirements elicitation methods and their concepts, not only actors in use case modeling but also use case, goals in goal-oriented approach, etc. And it can elicit security objectives and SF components besides threats. Furthermore, we adopt a method weaving technique based on a twin peak model and our method is more sophisticated rather than Ware's approach.

Liu et al. [5] and Mouratidis et al. [6] proposed the usage of the concept of soft goals in goal-oriented approaches such as i* and Secure Tropos. However, their approaches did not include the guidelines or the methods to assist in identifying threats and security objectives, and in refining security-related goals in a graph. Thus their approaches can be considered only as the detailed version of the step 6 of our approach.

The approaches using misuse cases [2] and anti-goals [12] may be helpful to identify threats and can correspond to the steps 2 and 3 of our approach. However, their supporting techniques of refining and decomposing use cases or goals are not so powerful to derive countermeasures after identifying threats. That is to say, any of them did not consider the powerful supports for the activities of the right-hand side of Figure 2.

Industries have tried to extract and structure knowledge included in CC so as to assist in eliciting security requirements. For example, a STF (specialist task force) in ETSI has worked on this aim since 2003 [13]. The structured and classified security knowledge that it has produced can be helpful to make our method more sophisticated, and our approach can be considered as a bridge between its outcomes and existing requirements elicitation methods.

6 Research Agenda

This paper proposes the technique to use CC as reusable knowledge within the existing requirements elicitation methods in order to elicit security requirements. And it helps the weave of requirements elicitation between a problem space and a solution one, following a twin-peak model. However, there are several unsolved research agenda to be tackled. Firstly, the attributes attached to assets and the inference rules should be explored and elaborated. They may depend on the adopted FRs elicitation methods. Secondly, as mentioned in section 4, the usage of the words appearing in CC would be promising to select the catalogued threats, security objectives and SF components. That is to say, we will investigate the application of ontological approaches, e.g. words as ontological concepts and the selections as ontological inference respectively. Thirdly, we also consider that the other types of knowledge sources for security requirements, e.g. various levels of security patterns [14,15], attack patterns of SAFE-T [16], Microsoft's STRIDE catalog [17], Security Ontology catalog such as NRL Ontology [18] and CVSS [19] could be integrated with our approach. Lastly, we should have more practical case studies to evaluate our technique.

References

1. Haley, C., Moffett, J., Nuseibeh, B.: Security Requirements Engineering: A Framework for Representation and Analysis. IEEE Trans. on Software Engineering 34(1), 133–153 (2008)
2. Sindre, G., Opdahl, A.: Eliciting Security Requirements with Misuse Cases. Requirements Engineering 10(1), 34–44 (2005)
3. McDermott, J., Fox, C.: Using Abuse Case Models for Security Requirements Analysis. In: Proc. of the 15th Annual Computer Security Applications Conference, pp. 55–64 (1999)
4. Firesmith, D.: Security Use Cases. Journal of Object Technology 2(3), 53–64 (2003)
5. Liu, L., Yu, E., Mylopoulos, J.: Security and Privacy Requirements Analysis with a Social Setting. In: Proc. of the 11th IEEE International Requirements Engineering Conference, pp. 151–161 (2003)
6. Mouratidis, H., Giorgini, P.: Secure Tropos: A Security-oriented Extension of the Tropos Methodology. International Journal of Software Engineering and Knowledge Engineering 17(2), 285–309 (2007)

7. Nuseibeh, B.: Weaving Together Requirements and Architectures. IEEE Computer 34(3), 115–117 (2001)
8. Official CC/CEM Versions - The Common Criteria Portal (2007), http://www.commoncriteriaportal.org/thecc.html
9. ECMA-271: Extended Commercially Oriented Functionality Class for Security Evaluation, E-COFC (1999), http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-271.pdf
10. Fernandez, E., Larrondo-Petrie, M., Sorgente, T., Vanhilst, N.: Methodology to Develop Secure Systems Using Patterns. In: Mouratidis, H., Giorgini, P. (eds.) Integrating Security And Software Engineering: Advances And Future Vision, pp. 107–126 (2006)
11. Ware, M., Bowles, J., Eastman, C.: Using the Common Criteria to Elicit Security Requirements with Use Cases. In: Proc. of the IEEE Southeast Conference, pp. 273–278 (2005)
12. van Lamsweerde, A.: Elaborating Security Requirements by Construction of Intentional Anti-Models. In: Proc. of 26th International Conference on Software Engineering (ICSE 2004), pp. 148–157 (2004)
13. TISPAN security: Adoption of Common Criteria in security evaluation (2003), http://portal.etsi.org
14. Heyman, T., Yskout, K., Scandariato, R., Joosen, W.: An Analysis of the Security Patterns Landscape. In: Proc. of the 3rd International Workshop on Software Engineering for Secure Systems (2007)
15. Yoshioka, N., Washizaki, H., Maruyama, K.: A Survey on Security Patterns. Progress in Informatics (5), 35–47 (2008)
16. Gegick, M., Williams, L.: Matching Attack Patterns to Security Vulnerabilities in Software-intensive Designs. ACM SIGSOFT Software Engineering Notes 30(4), 1–7 (2005)
17. Threat Modeling: Uncover Security Design Flaws Using The STRIDE Approach (2006), http://msdn.microsoft.com/en-us/magazine/cc163519.aspx
18. Kim, A., Luo, J., Kang, M.: Security ontology for annotating resources. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3761, pp. 1483–1499. Springer, Heidelberg (2005)
19. Mell, P., Scarfone, K., Romanosky, S.: A Complete Guide to the Common Vulnerability Scoring System Version 2.0 (2007), http://www.first.org/cvss/cvss-guide.html

Second International Workshop on the Model-Based Design of Trustworthy Health Information Systems MOTHIS 2008

Ruth Breu[1] and János Sztipanovits[2]

[1] Universität Innsbruck, Institut für Informatik, Techniker Straße 21a, 6020 Innsbruck, A
ruth.breu@uibk.ac.at

[2] Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA
janos.sztipanovits@vanderbilt.edu

Abstract. The objective of the MOTHIS workshop was to discuss model-based methods for the design of Health Information Systems (HIS) with a focus on trustworthiness, security and availability. While quality and affordability of health care delivery represents a major societal challenge in the 21st century the current situation in health care is still dominated by paper records and fragmented, error-prone approaches to service delivery. The workshop brought together computer scientists and medical experts to discuss research results in the development and application of model-based methods for representing, analyzing and integrating, architectures, privacy and security policies, computer security mechanisms, and human factors engineering.

1 Introduction

The Second International Workshop on the Model-Based Design of Trustworthy Health Information Systems (MOTHIS) took place on September 30th, 2008 in Toulouse, France. The objective of MOTHIS was to discuss model-based methods for the design of Health Information Systems (HIS) and to foster a community of experts in eHealth, software engineering and security.

While quality and affordability of health care delivery represents a major societal challenge in the 21st century the current situation in health care is still dominated by paper records and fragmented, error-prone approaches to service delivery. Model-based approaches provide a significant step towards the systematic analysis of requirements and high-level construction of solutions. Therefore, they are considered an important contribution to realize emerging interactive application scenarios and to support complex safety and security critical processes in health care.

Approximately fifteen people attended MOTHIS. The program consisted of one keynote address and two regular sessions. For the detailed program and the online proceedings, visit `http://qe-informatik.uibk.ac.at/mothis2008/`.

2 Workshop Summary

Alfred Winter, full professor for medical informatics at the Institute of Medical Informatics, Statistics and Epidemiology at the University of Leipzig, Germany, focused in this keynote address on the modeling and analysis of IT landscapes. Based on the 3LGM2, a meta model for modeling information systems at three layers of abstraction, quality aspects like data and service redundancy, and availability of IT services can be systematically analyzed. Winter showed case studies in which complex IT landscapes of hospitals have been modeled and analyzed.

S. Walderhaug et al., reported on experiences from model-driven development of homecare services in the context of the EU-IST project MPOWER. The goal of MPOWER is to define and implement an open platform to simplify and speed up the task of developing and deploying services for persons with cognitive disabilities and elderly. The contribution reported on the use of a DSL and a model-driven tool chain and came to the conclusion that the UML profiles used can improve modeling process performance and results.

Jason B. Martin et al use model-based techniques for guideline-driven clinical process support. Their approach is part of MICIS, a generic tool suite for designing, testing and deploying clinical information systems. The contribution showed the use of the framework for specifying and implementing guidelines to support decisions within workflows in health care. The approach comprises a modeling language and the application of techniques for model validation and verification. The approach has been applied to the management of sepsis in acute care settings at Vanderbilt Medical Center.

A. Strübing presented the paper of A. Winter et al. which introduces a formal definition of functional redundancy for health information systems. Based on the 3LGM2 meta model a key performance indicator for evaluating the redundancy rate of an IT landscape is defined. In addition, an algorithm for calculating non redundant system architectures supports the tool-supported model-based analysis and improvement of health information systems architectures.

J. Werner et al. addressed the gap between legal regulations (like complex data protection legislation in the HIPAA) and their enforcement in clinical information systems. The authors propose an approach to integrate formal logical representations of privacy policies with workflow models through a common semantic platform. It is shown that a model-based development approach can be leveraged to develop clinical information systems that comply with privacy legislation in a verifiable manner.

B. Katt and M. Hafner in their position paper sketched the core ideas of a framework for the modeling and enforcement of usage control-based privacy policies. The approach is based on meta models for the platform independent modeling of privacy policies which in a second step are transformed in platform specific models configuring a security architecture. Examples demonstrated the relevance of the approach for the health care domain.

Experiences from Model-Driven Development of Homecare Services: UML Profiles and Domain Models

Ståle Walderhaug[1,2], Erlend Stav[1], and Marius Mikalsen[1]

[1] SINTEF ICT, SP Andersens vei 15b, N-7465 Trondheim, Norway
[2] Department of Computer Science, University of Tromsø, N-9037 Tromsø, Norway
{stale.walderhaug,erlend.stav,marius.mikalsen}@sintef.no

Abstract. Model-driven development approaches such as Model Driven Architecture (MDA) have been proposed as the new paradigm for software development. The adoption of MDA is still low, partly because of the general-purpose modelling language being used. Domain specific modelling languages are being developed for technological and industrial domains to improve the expressiveness and effect of model-driven development techniques. The healthcare domain could benefit from these methodologies. In order to incorporate domain knowledge in a MDA process, information about workflows, artefacts and actors can be formalized in a UML profile and applied by MDA tools for design and development. This paper presents the work done on model-driven development of smart homecare services in the MPOWER project. Following an iterative approach, two UML profiles to support development of Service Oriented Architecture based homecare applications are proposed. Using homecare specific UML profiles indicate an improvement in the process for model-driven development of homecare services.

1 Introduction

Model-Driven Development (MDD) such as OMG's Model Driven Architecture (MDA) [1], has the potential to improve the quality of software systems. Quality attributes such as interoperability, reusability and appropriateness of software components and systems are main features of MDA. By using abstraction and advanced automation techniques, software artefacts are created from formal models that are represented using languages such as the Unified Modeling Language (UML) [2]. The core of the MDA process, and similar MDD processes, is to use formal models as the main development artefacts in the entire development process, from domain analysis to implementation, deployment and testing.

Domain specific modelling languages (DSML) have been proposed as a means to overcome many of the shortcomings with UML and MDA. The scientific knowledge about applying MDD techniques in design and development of healthcare information systems is scarce [3]. Creating DSMLs for the healthcare domain is a daunting task, and requires extensive investment of resources and time.

We set out to investigate how MDD with DSML support should be introduced and applied in a healthcare sub domain. In the MPOWER project [4], we have developed a framework for creating homecare software services using a model-driven approach. The framework defines a MDA toolchain which is a set of modelling, transformation and development tools that supports the complete MDA process as descibed in the MDA Guide [1]. A comprehensive model of actors and services in homecare along with the MDA toolchain for designing and implementing these domain specific web services has been developed and evaluated. This paper presents research results from the project with focus on:

1. What is the domain knowledge in homecare that can be used as assets in the MDA process?
2. Which knowledge can be included in a UML Profile for homecare services and how can this knowledge be utilized by developers?

The MPOWER toolchain, providing model traceability, model transformation and code generation, has been evaluated in the development of two proof-of-concept applications and is currently been redesigned with improved UML Profile support for the domain. Based on the experience from developing the MPOWER framework and proof-of-concept applications a conceptual domain model and UML profile for service oriented computing in the homecare domain is proposed and discussed.

The remainder of this paper is organized as follows. The next section describes the background for the work, including relations to and motivations for applying model driven development, domain specific modelling languages, and service oriented architecture. Then the applied method and main activities within the MPOWER project are described. The main results from each of the main activities are presented next, including conceptual domain models and our preliminary DSML approach based on two UML profiles. A discussion follows this, before we conclude the paper.

2 Background and Related Work

The work presented herein is a part of the EU-IST project MPOWER (contract no. 034707) and of an ongoing PhD thesis work by the main author. MPOWER is a user driven research and development project where the main goal is to create a framework for rapidly creating standards-based homecare services. The framework includes the definition of a toolchain which is being used in the development of two proof-of-concept applications targeting elderly and cognitive impaired people living at home.

MDD promises a potential to improve the quality of software systems and their development by using formal models as first class entities in the entire development process. When MDD is done properly, improvements in the design, development and maintenance processes can be achieved, in addition to improving the interoperability aspects and reducing the overall development costs.

Tuomainen et al argues that modelling helps the understanding of healthcare activities by being illustrative, identifying improvements, simulate organisational processes and individual activities in healthcare [5]. They compare three model centric approaches; MDA, Business Process Modelling with BPMN and BPEL and the HL7 development framework. They conclude that in order to realise their full potential these approaches require local and project specific adaptation. This paper explains such an adaptation for the homecare domain.

3 Methods

The main objective of the MPOWER project is to create a framework that facilitates rapid development of homecare services. To achieve this, it is imperative to acquire knowledge about the homecare domain, and make this domain knowledge available to actors involved in the system development processes. Due to the complexity of the healthcare domain, it was considered necessary to iterate between domain modelling and system design. To facilitate this interaction, the MPOWER project defined three main activities:

1. Capture domain knowledge from experts on aging/dementia, healthcare workers in the domain, family carers and patients.
2. Specify a MDA toolchain that support documentation of system requirements, modelling of design and development of services. Moreover, the toolchain must be evaluated in terms of usability and usefulness/performance by implementing two Proof-of-Concept Applications (PoCA). The results are a MDA toolchain with evaluation reports on developer acceptance and technical qualities
3. Design a DSML that incorporates the domain knowledge from task 1 and MDA toolchain experience from task 2. The result will be one or more UML profiles that can be used with a revised MDA toolchain.

3.1 Activity 1: Capture Domain Knowledge

The MPOWER project focuses on smart homecare solutions for elderly and cognitive impaired people. The domain models for the work being presented in this paper can be seen from two different viewpoints:

1. The Homecare viewpoint: this viewpoint focuses on organizational aspects of homecare as well as the main stakeholders (people and systems) involved. This model is the result of a comprehensive process involving a total of 140 domain stakeholders such as domain experts, professional caregivers, family carers and patients [6].
2. The Homecare SOA viewpoint: this viewpoint focuses on the main system components and their relationships in terms of the principles of Service Oriented Architecture design. Important assets for this model are the design principles given by Erl [7], and SOA4HL7 methodology [8]. The structure and semantics of the domain model is supported by the SOA reference architecture from IBM along with the IBM UML profile for software services [9].

3.2 Activity 2: Designing a Toolchain for MDD in Homecare

To have a formal way of specifying the domain models, proper modelling tools are needed. In the beginning of the MPOWER project, a set of tools were specified as the MPOWER toolchain to be used by all involved personnel for conceptual modelling, requirements specification, analysis, system design, system development, deployment and testing. The process used for selection of tools matches the recommendation given by Staron [10], page 240: "The process [of creating domain models] should be tool independent. The independence should be supported by using technologies that are open and unbounded, but at the same time supported by more than one tool." In MPOWER, Microsoft Word 2003 was used for describing user scenarios and Enterprise Architect (EA) V6.5 from Sparx Systems used for UML modelling of use cases and services, model transformation and code generation of WSDL code. THE IBM's UML 2.0 profile for Software Services was applied during modelling the services to structure models and stereotype coreelements. Available from IBM [9]. Finally, NetBeans V6.0 was used as Java IDE for generating service skeletons and implementing the services. `http://www.netbeans.org`

The described toolchain was used from the start of the project with only minor modifications such as EA upgrades and bug fixes. The two PoCAs were developed using the toolchain and the performance of the toolchain, were investigated from two perspectives: 1) Developer acceptance of the MPOWER toolchain: using the Technology Acceptance Model with two additional factors, as reported in [11], and 2) Technical review: Weekly scrum and quarterly technical meetings with workshop sessions on how to improve the toolchain.

3.3 Activity 3: Refine the MPOWER Toolchain and Develop a DSML

The UML standard allows for the creation of a DSML in two ways: 1) Creating a new language based on Meta Object Facility [12], or 2) extending UML through the use of UML Profiles. As discussed by Selic, the latter will often be the most practical and cost-effective solution [13], and is also the chosen method for this paper. By using UML profiles to create a DSML, the semantics and syntax of UML can be inherited, and powerful UML/MDA tools can be used with the profile for software development.

The UML Profile standard [2], outlines several reasons for creating a DSML from UML. The most pertinent reasons for the challenges addressed in this paper are:

– Terminology adapted to the healthcare domain
– Add information that can be used during transformation
– Add constraints that restrict the way you can use the metamodel

There is not much knowledge in the scientific community about best practices for creating DSMLs with UML [13] [14]. In a paper from 2007, Selic summarized the basic steps for creating a DSML in terms of UML in [13]: 1) Create a conceptual domain model that includes the essential concepts of the domain, the relationships between the concepts, the constraints that govern the use of the concepts. A selection of UML models from Activity 1 makes up the conceptual domain model, 2) Map domain model to a UML profile, refining the core UML specification with stereotypes, tagged values and constraints.

The process of creating a domain specific UML profile is not straightforward, since the level of abstraction and the intended use of the profile play an important role for the definition of the profile elements. This challenge is tackled with experience from the design of the MPOWER toolchain and development of two MPOWER PoCAs for the homecare domain. To identify and model the elements of a UML profile is an iterative process. To guide this process, the Staron's guidelines for defining good stereotypes using a classification schema [10], is used.

4 Results

The results presented in this section are based on the work carried out in the MPOWER project from October 2006 to June 2008.

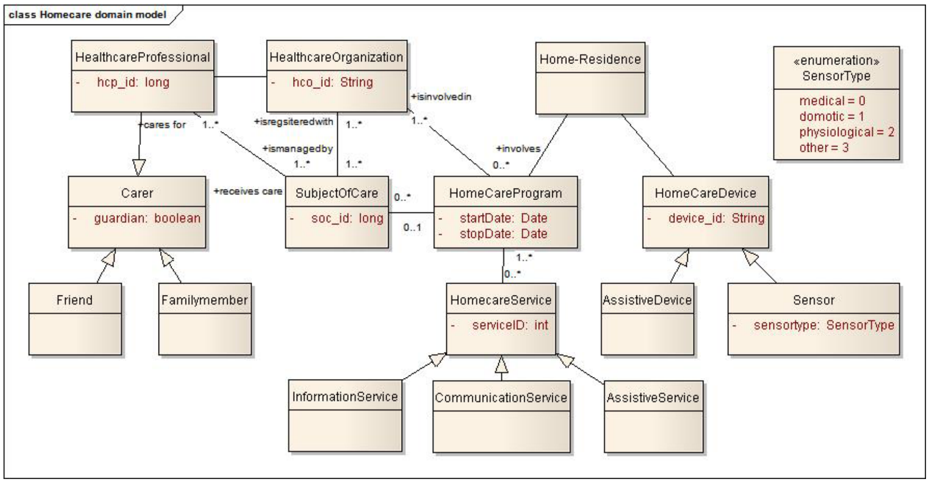


Fig. 1. Diagram showing the main concepts in a smart homecare domain

4.1 Activity 1: Conceptual Domain Models

The domain models were developed in several iterations from October 2006 to September 2007. Figure 1 shows the main concepts from a homecare viewpoint.

To keep the model at an abstract level and not overpopulate it with unnecessary details, most attributes on the classes are hidden. The main classes and relationships are:

Table 1. The main classes and relationships in the homecare domain

Sterotype	Comment
Subject of Care (SoC)	person receiving care through a homecare program. The SoC has a unique identifier that is managed by the assigned healthcare organization. A SoC must be associated with at least one healthcare professional
Homecare Program	a class comprising the services, devices and healthcare organizations involved in providing homecare service to a SoC.
Carer	an individual that is a part of the family, a healthcare professional or a friend. All HealthcareProfessionals must be associated with a HealthcareOrganization.
Healthcare Organization	an organisation that is directly involved in the provision of care to a SoC.
Homecare Service	a service provided to the SoC through a Homecare program. Three core types of services have been identified: information service (e.g., calendar, medication list), communication services (e.g. SMS, email), and assistive service (e.g. indoor location service, heating control, burglar alarm, oven control).

Concepts in the model are aligned with the concepts presented in Continuity of Care (CONTSYS) standard [15], and service categories from [16]. Most concepts are also available in the HL7 RIM, but CONTSYS is more specific than HL7. These resources were found useful in selecting an appropriate abstraction level and structure in the domain model. The complete models of actors and services are presented in [6]. Figure 2 shows the main components, stereotyped with the five layers of the IBM SOA reference architecture.

4.2 Activity 2: The MPOWER Toolchain

The experiences from using the described toolchain for development of the services that form the PoCAs are grouped into developers' subjective experience and technical experience. The first group entails perceived characteristics such as the factors described by the Technology Acceptance Model [17]. The results from a developer evaluation of the MPOWER toolchain is presented in [11], and concludes that perceived ease of use and perceived usefulness are factors that affect the developers' adoption of MDA. It was also found that traceability between artefacts in the development process was useful. A major drawback with the evaluated toolchain was found to be the incomplete code generation features. A technical review of the MPOWER toolchain revealed that 1) WSDL model transformation incomplete: it was necessary to customize the transformation template for WSDL models, 2) WSDL code generation had errors: the built-in transformations in the EA tool generated some errors that had to be changed manually, e.g. using "type" references instead of "element" references in message definitions, and 3) Performance problems using HL7: the import of HL7 message types into WSDL resulted in tool crashes because of memory allocation

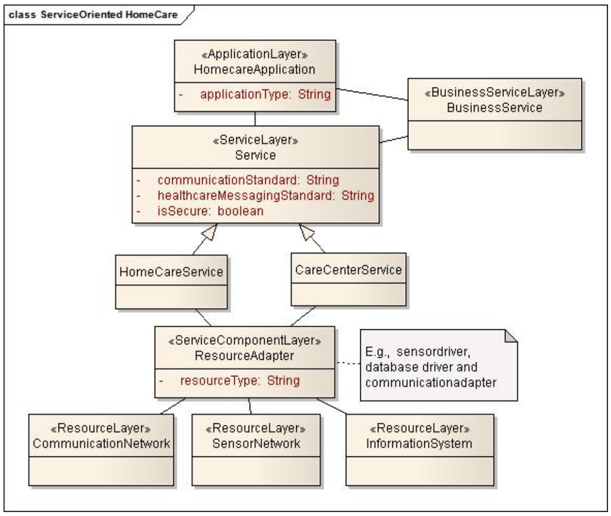

class ServiceOriented HomeCare
«ApplicationLayer» HomecareApplication
- applicationType: String
«BusinessServiceLayer» BusinessService
«ServiceLayer» Service
- communicationStandard: String
- healthcareMessagingStandard: String
- isSecure: boolean
HomeCareService
CareCenterService
«ServiceComponentLayer» ResourceAdapter
- resourceType: String
E.g., sensordriver, database driver and communicationadapter
«ResourceLayer» CommunicationNetwork
«ResourceLayer» SensorNetwork
«ResourceLayer» InformationSystem

Fig. 2. The Service-oriented view on a typical homecare environment

problems. Recursive import of HL7 xml schema (xsd) defintions were not handled by the WSImport tools in Netbeans.

From the experience with the MPOWER toolchain, a set of new features were proposed. The developers would like more support in generating the implementation of the services – repetitive code (e.g. for DB management, handling of security, return status), and support for object/relational persistence service, such as generation of Hibernate mappings for the information elements declared in the message definitions of the WSDLs. These features may impact the design of a DSML as they could require domain specific information to be incorporated into the models during the service design.

4.3 Activity 3: Refined Toolchain - Mapping of Domain Concepts to DSML - UML Profile

The process of defining a UML Profile for SOA in homecare use concepts from the conceptual domain models and experience from toolchain and PoCA development in an iterative approach. This section presents the preliminary results from Activity 3 after the first iteration (January-June 2008). Activity 3 is carried out by a core team of three researchers. The profiles were updated in three main revisions: initial version for the start of the development, second version after first version of services and the third version after the first iteration of the PoCA development. The changes between version one and two were significant, whereas only tags and minor adjustments to relationships were done for version 3.

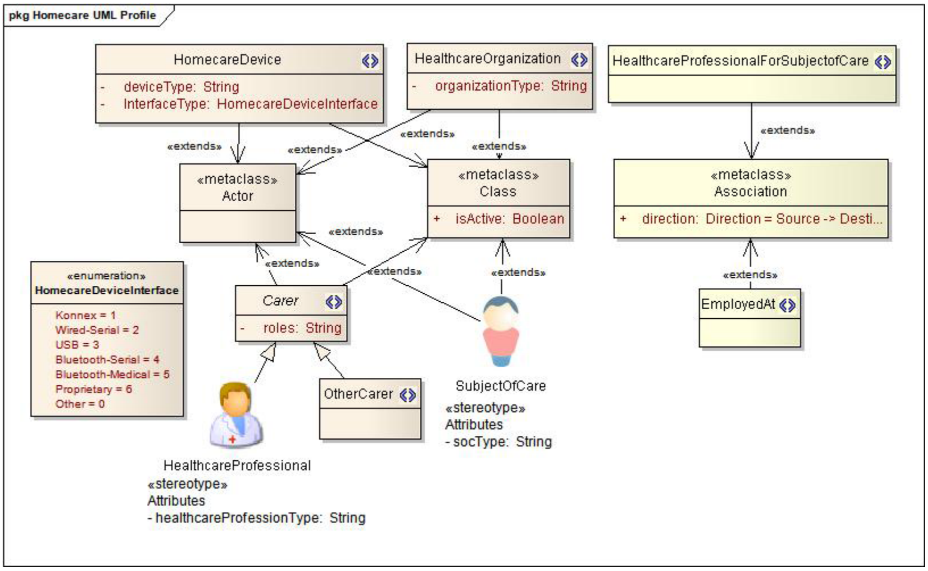


Fig. 3. First version of Homecare UML Profile

Two UML profiles are proposed, the Homecare UML Profile and the SOA Homecare UML Profile. The profiles can be used separately or together in a MDA development project, depending on how the profile elements are utilized in the development process.

Figure 3 shows the Homecare UML profile. The profile elements are mainly derived from the homecare conceptual model (figure 1). The tagged values on the stereotypes were identified based on experience from the MDA toolchain work. The mapping of concepts to the UML profile was also guided by the CONTSYS standard [15].

All the stereotypes in the Homecare UML Profile falls into the category "Virtual Metamodel Extension, restrictive" defined by Staron in [10]. These are stereotypes that reuse the semantics of the metaclasses (e.g. Actor and Class). Often they must be used with other stereotypes, making the stereotyped model element more precise and may also add a new icon to the concrete syntax to familiarize the model presentation (e.g. icons on HealthcareProfessional and SubjectOfCare).

The SOA Homecare UML profile enables developers to create precise models of SOA-based homecare systems. Figure 4 shows the core elements of the SOA Homecare UML profile.

Table 2. Table describing the proposed stereotypes in and tagged values in the Homecare UML profile

Sterotype	Comment
Subject Of Care	Subject of care (SoC) is defined in CONTSYS as ”person seeking to receive, receiving, or having received health care” [15]. Used to decorate SoC modelling elements and to add information about the SoC that can be used during model transformation or code generation. SoC_type: describes can be used to describe different types of SoC according to e.g., national specific patient classifications.
Carer	A stereotype that should be used on all modelling elements representing an individual that provides care, professionals as well as non-professional caregivers. roles: can be used to set the default role, e.g. in terms of security, for the instances of classes marked with this stereotype
Healthcare Professional (HcP)	Defined in CONTSYS as “person authorised by a nationally defined mechanism to be involved in the direct provision of certain health care activities” [15]. Should be used to mark all modelling elements of type Class/Actor that are representing individuals that fit this definition. The roles attribute is inherited from Carer.
Other Carer	Defined in CONTSYS as “person providing assistance for activities of daily living or social support”. This stereotype should be used to mark modelling elements of type Class/Actor that are representing individuals such as family members, friends and other carers employed by non-healthcare organizations such as home services and security services. The roles attribute is inherited from Carer
Healthcare Organization (HcO)	Defined in CONTSYS as “organisation involved in the direct provision of health care” [15]. This stereotype should be used to mark all modelling elements of type Class/Actor that represents organisations that fits the CONTSYS definition. orgainsationType: is used to describe the type of organisation according to speciality levels, private versus public or other national classifications.
Homecare Device	Generic homecare device stereotype to be used on modelling elements of type Class/Actor. The stereotype can be useful for design-time checking of interoperability and interconnectivity of devices in a homecare system. deviceType: describes the type of this device interfaceType: describes the kind of interface used to connect to this device. In the UML profile, the HomecareDeviceInterface enumeration is defined based on the experience in the MPOWER project, but can be refined to fit other technologies.

Table 2. (*continued*)

Sterotype	Comment
Healthcare Professional For Subject Of Care	A stereotype that is used to mark an association between a HealthcareProfessional and a SubjectOfCare. The stereotype can be used to ensure that a SubjectOfCare is associated with at least one HealthcareProfessional.
EmployedAt	A stereotype that is used to mark an association between a HealthcareProfessional and a HealthcareOrganisation. The stereotype can be used to check that all HealthcareProfessional "types" are employed at a HealthcareOrganisation "type".

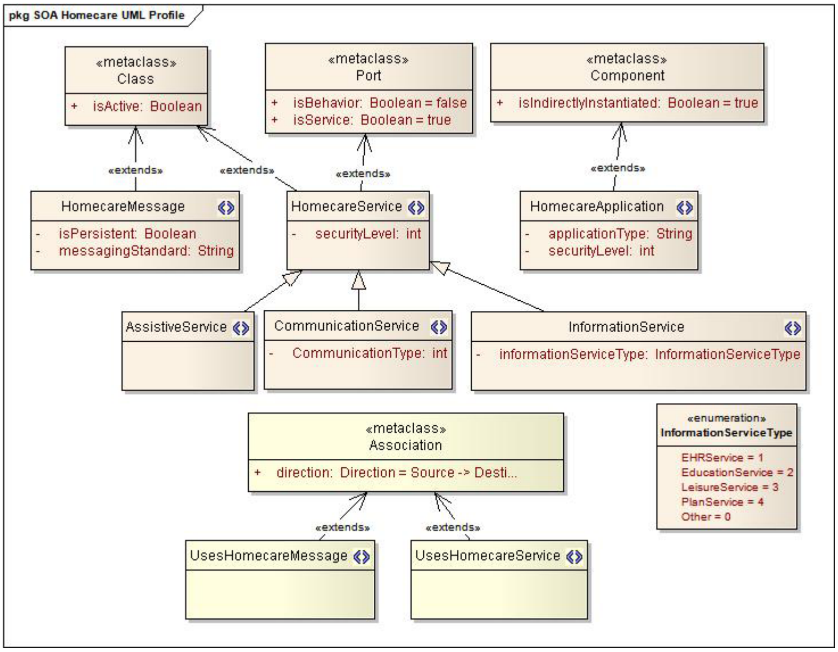


Fig. 4. The Homecare SOA UML Profile diagram

All stereotypes in the Homecare SOA UML profiles falls into the category "Code generation, restrictive" defined by Staron [10]. These are stereotypes that extend the base metaclass (e.g., Class and Port) with some properties to increase the precision of the semantics and restrict the usage. Each stereotype is described in more detail in the table below.

Table 3. List of stereotypes and tagged values in the SOA Homecare UML profile

Sterotype	Comment
Homecare Application	The stereotype adds two properties to the modelling elements: securityLevel: this describes the security level of the application. This can be used to check that a user (i.e., service, component, application) of the application must have at least the same access level in order to be allowed to use the service. applicationType: this describe the type of application this is, e.g. in terms of deployment configurations.
Homecare Service	A service which is used in the homecare environment. The IBM Service Profile should be used in combination with this stereotype. securityLevel: this describes the security level of the service. This allows for specification of the security requirements and rights for a modelling element that can be utilized during code generation. In the next version of the UML Profile, this stereotype will be updated with more security tags addressing service-service authorization and information encryption.
Homecare Message	To denote elements that are messages used in interactions of homecare services and applications. isPersistent: indicates whether the message data is stored in a database or not. This can be used for creating Hibernate mapping code and database schema. messagingStandard: the standard to which this message belongs. Can be used both for code generation, ensuring correct libraries are present, and for checking conformance with the standard.
Assistive Service	A type of homecare service that provides assistive functionality in the homecare system. Derived from Stefanov's classification for smart house services for elderly and cognitive disabled [16].
Information Service	An information service which will be used by stakeholders in a homecare setting. serviceType: defines the type of information service this service belongs to. An enumeration is proposed based on the experience in the MPOWER project.
Uses Homecare Message	A stereotype that mark associations between a HomecareService and a HomecareMessage. The stereotype can be used to generate traceability information that can again be used when messaging standards are being updated or changed.
Uses Homecare Service	A stereotype that mark associations between a HomecareApplication and a HomecareService. The stereotype can be used to generate traceability information that again can be used when a homecare service is being updated or changed.

5 Discussion

The work presented in this article is a part of an initiative to develop a MDD framework for healthcare, focusing on homecare services in the first phase. It is considered imperative to incorporate domain knowledge into the framework and make this knowledge readily available for architects and developers in all development phases. This paper presents the results from creating a domain specific modelling language for homecare using UML Profiles.

Capturing the conceptual domain knowledge for homecare, or any other healthcare sub-domain, is a daunting task. Many stakeholders are involved, as well as a plethora of information systems, involving many different coding standards and vocabularies. These factors, in addition to legislative factors and organisational aspects, make modelling of reusable healthcare domain models difficult, but not impossible. To succeed in creating a useful model-driven software development process, it is important to choose the right level of abstraction and in divide the healthcare application areas into well defined sub-domains.

The homecare domain model shown in figure 1 shows the most important actors and relationships between them. The model would fit for modelling most homecare solutions, is aligned with the CONTSYS standard [15], and includes the main classifications from Stefanov's paper on smart house technologies for elderly [16]. The model is the result of a comprehensive domain analysis process where 140 domain stakeholders from four European countries were involved in improving the validity of the model [6]. If new concepts are developed for the domain, these can be added as an extension to the existing homecare model, without compromising the original model and the related UML profiles.

The Homecare SOA model is based on the domain investigation from the MPOWER project, in addition to the IBM SOA reference model and IBM UML profile for software services [9]. The Homecare SOA model provides information about deployment of services and possible configuration and information sources. The model is on an abstract level, and could in certain cases be refined with details about security platforms and network connectivity details. Such domain knowledge could be useful in planning the distribution of services and integration with existing resources, but will also make the model less suitable for reuse across different healthcare enterprises and nations.

To make the domain knowledge readily available as assets in the development process, UML profiles were chosen, inline with the recommendations by Selic [13]. UML profiles builds upon the syntax and semantics of UML, and most UML tools support profiles. This is an imperative advantage, enabling developers to use their favourite UML tool for design and development. The process of selecting domain concepts to include as stereotypes, tagged values or constraints in a UML model, requires knowledge about model-driven development, but also experience from modelling systems in the domain in question. Experience from the development of a MDA toolchain (Activity 2) provided information about which target software artefacts that should be generated from the models and which models and diagrams that should be applied for achieving this. This information was of utmost importance when choosing the metaclass extensions for the elements in the UML profiles.

UML profiles are used to customize the modelling language to include familiar concepts that enables more effective and precise system design and implementation. Stereotypes can be used for several purposes, as discussed by Staron [10]. The result from the mapping of domain concepts to a DSML (Activity 3) showed that all stereotypes in the Homecare UML profile are classified as Virtual Metamodel Extensions. This implies that this profile is mainly used to increase the expressiveness of the modelling language when designing systems for homecare. A "virtual metamodel extension, restrictive" stereotype adds a domain specific icon such as a picture of a nurse to the modelling element, together with a well known domain specific label such as HealthcareProfessional.

The Homecare SOA UML Profile includes elements from the "Code generation, restrictive" category. These stereotypes can improve code generation by providing domain information so that code generation scripts can create high-quality code. In this paper code generation was restricted to WSDL and Hibernate code, but other software artefacts can be generated from the domain information in a UML profile. Test cases, error-checking code, security policy verification, and result validation may also be generated.

The two proposed UML profiles can be used on the same models to provide different "views". In addition, the IBM Software Service UML profile should be used to complement the service design models for SOA Homecare systems.

6 Concluding Remarks

Model-driven development approaches can be improved by extending the modelling language with domain specific concepts. UML Profiles can be used as a mechanism for toolchains based on OMG's MDA and UML standards.

The UML Profiles must provide information that can improve the design and/or code generation processes. The two profiles proposed in this paper are based on solid work on capturing homecare domain knowledge and experience from developing homecare SOA systems using MDA. Though the profiles are still undergoing updates and improvements, they can improve modelling process performance and results.

It was found useful to have a development activity in parallel with specification of the conceptual domain and profiles. The experience from the development activity gave valuable input to the mapping of concepts to DSML processes. This finding extends the approaches to DSML development in [13] [14] [18].

The MPOWER Toolchain will be evaluated by university students in 2008. In 2009, an experiement measuring the subjective improvements (perceived characteristics) and objective improvements (e.g., reduction of errors, time spent for development) will be conducted with 20 developers from the healthcare domain.

References

1. Miller, J., Mukerji, J.: MDA Guide Version 1.0.1. In: Miller, J., Mukerji, J. (eds.) Object Management Group (OMG), pp. 1–62 (2003)

2. Object Management Group (OMG), UML 2.1.2 Superstructure and Infrastructure, Object Management Group (OMG) (2007)
3. Mohagheghi, P., Dehlen, V.: Where Is the Proof?-A Review of Experiences from Applying MDE in Industry. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 432–443. Springer, Heidelberg (2008)
4. MPOWER Consortium (2006) (cited November 15, 2006), http://www.sintef.no/mpower
5. Tuomainen, M., et al.: Model-centric approaches for the development of health information systems. In: Medinfo. 2007 Brisbane, Australia (2007)
6. Walderhaug, S., Stav, E., Mikalsen, M.: Reusing models of actors and services in smart homecare to improve sustainability. In: Medical Informatics Europe 2008, Gothenburg, Sweden. IOS Press, Amsterdam (2008)
7. Erl, T.: Service-Oriented Architecture Concepts. In: Erl, T. (ed.) Technology, and Design, Crawfordswille, Indiana, USA. The Prentice Hall Service-Oriented Computing Series. Prentice Hall, Englewood Cliffs (2006)
8. Honey, A., Lund, B.: Service Oriented Architecture and HL7 v3: Methodology, HL7 Service Oriented Architecture Special Interest Group (SOA SIG). p. 79 (2006)
9. Johnston, S.: UML 2.0 Profile for Software Services (2005) (cited November 15, 2008), http://www.ibm.com/developerworks/rational/library/05/419_soa/
10. Staron, M.: Improving modeling with UML by stereotype-based language customization. In: School of Engineering, p. 270. Blekinge Institute of Technology, Blekinge (2005)
11. Walderhaug, S., et al.: Factors affecting developers' use of MDSD in the Healthcare Domain: Evaluation from the MPOWER Project. In: From code-centric to model-centric develpoment, Workshop at European Conference on Model-Driven Architecture. European Software Instiitiute, Berlin (2008)
12. Object Management Group (OMG), MOF 2.0 / XMI Mapping Specification, v2.1 (2005)
13. Selic, B.: A Systematic Approach to Domain-Specific Language Design Using UML. In: 10th IEEE ISORC (2007)
14. Lagarde, F., et al.: Improving uml profile design practices by leveraging conceptual domain models. In: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pp. 445–448 (2007)
15. CEN TC251, EN 13940-1: Health Informatics - System of Concepts to Support Continuity of Care - Part 1: Basic Consepts. European Committee for Standardization, p. 105 (2006)
16. Stefanov, D.H., Bien, Z., Bang, W.-C.: The smart house for older persons and persons with physical disabilities: structure, technology arrangements, and perspectives. IEEE transactions on neural systems and rehabilitation engineering 12(2), 228–250 (2004)
17. Davis, F.D., Bagozzi, R.P., Warshaw, P.R.: User Acceptance of Computer Technology: A Comparison of Two Theoretical Models. Management Science 35(8), 982–1003 (1989)
18. Fuentes-Fernández, L., Vallecillo-Moreno, A.: An Introduction to UML Profiles. UML and Model Engineering 5(1) (2004)

Ontology-Based Assessment of Functional Redundancy in Health Information Systems

Alfred Winter[1], Alexander Strübing[1], Lutz Ißler[4], Birgit Brigl[2], and Reinhold Haux[3]

[1] University of Leipzig, Institute of Medical Informatics, Statistics and Epidem., Germany
[2] Dr. Birgit Brigl Krankenhaus-IT Management Beratung, Friedrichsdorf, Germany
[3] Peter L. Reichertz Institute for Medical Informatics of the University of Braunschweig - Institute for Technology and of Hannover Medical School, Germany
[4] Systemantics, Aachen, Germany

Abstract. The paper introduces a formal definition of functional redundancy to determine non-redundant health information system architectures, in order to support information management of, in particular, hospital information systems. We specify an ontology, which is linked to the Three-Layer Graph-Based Meta Model (3LGM^2) and based on enterprise functions and application systems of (health) information systems. A so called functional redundancy rate (FRR) is introduced and elucidated by an example. An algorithm for calculating non redundant health information system architectures is presented. Functional redundancy is a key performance indicator for the quality and efficiency of (health) information systems. With FRR it can now be formally described and quantitatively analyzed. Using 3LGM^2 based models of information systems, the calculation of FRR does not need further efforts.

Keywords: Health information systems, hospital information systems, information management architectural models, functional redundancy.

1 Introduction

Information management for health information systems has become a crucial and significant task, in particular for hospitals but also for 'trans-institutional' regional and national health care settings [1-3]. Assessing the quality and efficiency of health care institutions' information systems is an important field in research and practice of medical informatics [4, 5]. However, there is still a lack of easy to understand and likewise relevant evaluation criteria, which can be accurately defined and thus formally described. Such formal descriptions provide the option to immediately derive these criteria from architectural specifications of health information systems, and so to make them well suitable for the practice of information management. One of those criteria is functional redundancy ([6], pp. 170 and 233). Most information managers may have a certain feeling for redundancy of functional support in the information system they manage. But providing precise information regarding redundancy for decisions concerning the information system's architecture and investments still needs to be solved.

The aim of our research is to introduce a formal definition of functional redundancy and, to calculate a health information system's functional redundancy rate (FRR) (section 3) as well as to outline an algorithm for calculating non redundant health information system architectures (section 5). Before, we need to define an ontology [7] for describing functional redundancy in (health) information systems (section 2). On that basis we want to support information managers to find answers to the following questions:

- What application systems in my information systems can be shut down without loss of functionality?
- Do I have unnecessary costs because different users in my institution use different application systems in order to support the same enterprise function? What are the critical enterprise functions and what application system's usage should be prevented?

Please note that we are using the term information system in a rather comprehensive manner. An institution's (e.g. a hospital's) information system, is that socio-technical subsystem of the institution, which comprises all information processing actions as well as the associated human or technical actors in their respective information processing role [8]. The basic model, introduced in section 2, is closely linked to the Three-Layer Graph-Based Meta Model (3LGM^2) [9] serving as a domain ontology for the field of information systems [7].

2 An Ontological Foundation for Assessing Functional Redundancy in Information Systems

Describing and calculating functional redundancy in an information system first of all requires a model of the information system. To guarantee that the assessment of functional redundancy can be applied to the variety of existing modelling techniques, such a model should be based on an ontology for the description of information systems. To our knowledge, such ontology does not exist yet. But we identified two terms that are used in the most common modelling approaches, namely enterprise functions and application systems. Enterprise functions can be considered a directive for human or machine action and a duty arising from an enterprise's mission and goals. For example, "clinical admission", "radiotherapy", or "care planning" may be enterprise functions. Within the computer-supported part of an information system, the tools used to support the execution of enterprise functions can be described as application systems being installations of application software products on computers. Application systems may have a local database to store data and interfaces for communication.

Functional redundancy deals with the adequate relationship between tasks to be done, i.e. enterprise functions, and tools to support these tasks. Using these terms we can model this support relationship by a matrix *SUP*.

Let $\underline{EF}$ be a set of enterprise functions and $\underline{AS}$ a set of application systems.

$$\underline{EF} := \{EF_1,...,EF_P\}, \quad P > 0 \tag{1}$$

$$\underline{AS} := \{AS_1,...,AS_N\}, \quad N > 0 \tag{2}$$

The two-dimensional matrix *SUP* describing the relationship between tasks and tools mentioned above is defined as

$$SUP := \left(sup_{p,n}\right)_{p=1\ldots P, n=1\ldots N} \text{ with}$$

$$sup_{p,n} = \begin{cases} 1 \textit{ if function } ef_p \textit{ is suported by application system } as_n \\ 0 \textit{ else} \end{cases} \qquad (3)$$

Example (part 1)

Suppose a set of enterprise functions EF:={*A*,*B*,*C*,*D*,*E*,*F*,*G*} where the letters represent enterprise functions as follows: *A* for "clinical admission", *B* for "administrative admission (inpatients)", *C* for "administrative admission (outpatients)", *D* for "radiotherapy", *E* for "decision making", *F* for "patient information", and *G* for "care planning". Additionally, suppose a set of application systems AS:={*1*,*2*,*3*,*4*,*5*,*6*,*7*,*8*,*9*} where the numbers represent application systems as follows: *1* for "CareMgmtSys", *2* for "PatientAdministrationSystem(ADT)", *3* for "DepartmentalSystemPsychology", *4* for "DepartmentalSystemRadiotherapy", *5* for "KnowledgeService", *6* for "DiabetesTrainer", *7* for "ClinicalPathwaySys", *8* for "TherapyPlaner", and *9* for "TherapyAdvisor".

Table 1. The matrix *SUP* for EF and AS is illustrated in figure 1

		application systems n=1,…,9								
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
enterprise functions p=1,…,7	**A**	**1**	0	0	0	0	0	0	0	0
	B	0	**1**	0	0	0	0	0	0	0
	C	**1**	**1**	**1**	**1**	0	0	0	0	0
	D	0	0	0	**1**	0	0	0	0	0
	E	0	0	0	0	**1**	0	**1**	0	0
	F	0	0	0	0	0	**1**	0	0	0
	G	0	0	0	0	**1**	0	**1**	**1**	**1**

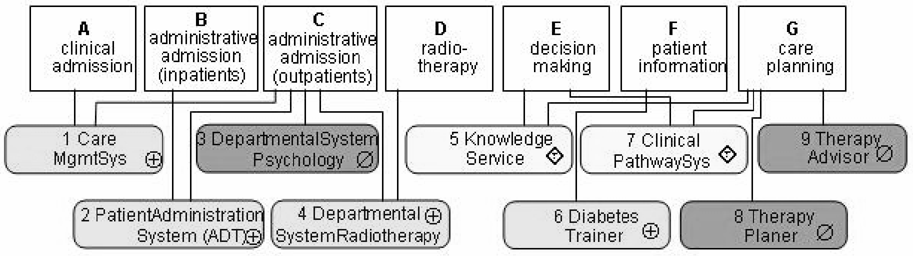


Fig. 1. Matrix *SUP*: rectangles denote enterprise functions, rounded rectangles denote application systems, and connecting lines illustrate a "1" in the respective position of the matrix, i.e. that a certain enterprise function is supported by a certain application system. E.g. enterprise function E "decision making" can be supported by application system *5* "KnowledgeService" or *7* "ClinicalPathwaySys" alternatively. The meaning of the different hatchings and the ⊕, ∅ and ⑨ -signs will be explained in section 4.3.

3 A Measure for Functional Redundancy

Functional redundancy is a characteristic of information systems which should be addressed by information management. In order to reduce complexity of the information system it is interesting to know which application systems could be omitted without loss of functionality, i.e. without hindering the execution of any enterprise function. Before we define a measure for functional redundancy we want to explain, how we can detect redundant support of enterprise functions by application systems.

3.1 Redundant Support of Enterprise Functions

For every enterprise function $ef \in \underline{EF}$ we can easily calculate

$$isup_p := \sum_{n=1}^{N} sup_{p,n} \tag{4}$$

For every *p*, $isup_p$ denotes the number of application systems actually supporting the individual enterprise function ef_p; we call it its individual degree of support by application systems. Every $isup_p > 1$ may be an indicator that some application systems are dispensable, with $isup_p - 1$ indicating the number of possibly superfluous systems. However, this number needs a careful investigation because some of the apparently superfluous application systems may be necessary for other enterprise functions. Obviously, measuring functional redundancy in a way, which is supportive for information management, needs a measure which takes these interrelationships into account.

Example (part 2)
Continuing part 1 of our example we can easily calculate the $isup_p$ as shown in table 2.

Table 2. $isup_p$

p	ef_p	$isup_p$
1	A	1
2	B	1
3	C	4
4	D	1
5	E	2
6	F	1
7	G	4

The value of $isup_3 = 4$ indicates that perhaps there are three superfluous application systems supporting *C* (“administrative admission (outpatients)”). But detailed analysis shows that the application systems *1* (“CareMgmtSys”), *2* (“PatientAdministrationSystem(ADT)”) and *4* (“DepartmentalSystemRadiotherapy”) cannot be omitted, because they are needed for the functions *A* (“clinical admission”), *B* (“administrative admission (inpatients)”) and *D* (“radiotherapy”). However, application system *3*

("DepartmentalSystemPsychology") is a good candidate for being removed from the information system because function *C* as the only function it supports is also supported by application systems *1*, *2*, and *4*. A measure of redundancy should therefore correctly indicate that considering the enterprise function *C* ("administrative admission (outpatients)") only one application system could be omitted (namely DepartmentalSystemPsychology).

3.2 A Measure for Functional Redundancy for Information Management

We now want to introduce a measure for functional redundancy for information management as a key performance indicator, denoting the percentage of application systems in a given information system, which could actually be shut down and omitted without loss of support of any enterprise function. First we have to check, whether particular application systems can be omitted or not, given $\underline{EF}$, $\underline{AS}$ and *SUP*. With the notions introduced earlier, the challenge is to calculate a minimal subset $\underline{AS}^{\min} \subseteq \underline{AS}$ of application systems which guarantees that all functions are supported and that there are no superfluous application systems in use. Each set $\underline{AS}^{\min}$ we call a "minimal functionally non-redundant set of application systems". In general, there is more than one such set $\underline{AS}^{\min}$ for a given information system, i. e. there is more than one way to cut down the functional redundancy in an information system. In the real setting of the information system of the Leipzig University Medical Center we actually found several hundreds of minimal functionally non-redundant sets of application systems.

Let us describe any subset $\underline{AS}' \subseteq \underline{AS}$ of application systems being actually in use by a vector $\overrightarrow{USE}$, indicating whether application systems are member of the subset $\underline{AS}'$ or not.

$$\overrightarrow{USE} := (use_n)_{n=1...N} \qquad \text{with} \qquad use_n = \begin{cases} 1\, if\ as_n \in \underline{AS}' \\ 0\, else \end{cases} \tag{5}$$

Hence $\underline{AS}^{\min}$ can be described by $\overrightarrow{USE}^{\min} := \left(use_n^{\min}\right)_{n=1...N}$ and $use_n^{\min} \in \{0,1\}$. Given what application systems are in use, i.e. given the respective vector *USE*, we can calculate the individual degree of support for all enterprise functions as well as:

$$\overrightarrow{ISUP}^T = SUP * \overrightarrow{USE} \qquad \text{with} \qquad \overrightarrow{ISUP} = \left(isup_p\right)_{p=1...P} \tag{6}$$

As stated above, we want that despite of some application systems being not in use, every function is supported by at least one application system. We introduce a vector $\vec{e}$ of length *P* containing only "1":

$$\vec{e} = \left(e_p\right)_{p=1...P} \qquad \text{with} \qquad e_p := 1,\ p = 1...P \tag{7}$$

Now we can state the first postulation:

(P1) For every vector $\overrightarrow{USE}$ which is as a candidate for being considered as a possible reduced set of application systems, the following constraint holds:

$$SUP * \overrightarrow{USE} \geq \vec{e}^{\,T} \tag{8}$$

Second we want to have as few application systems in use as possible. We introduce a vector $\vec{c}$ of length N containing only "-1":

$$\vec{c} = \left(c_n\right)_{n=1...N} \quad \text{with} \quad c_n := -1, n = 1...N \tag{9}$$

This leads to the second postulation:

(P2)
$$\vec{c}^{\,T} * \overrightarrow{USE} \rightarrow \max \tag{10}$$

Since *SUP* is a matrix of zeroes and ones, we have a pure 0-1 linear programming problem. This problem is well known in literature as the "set covering problem" [10]. Corresponding to our statement that there will be more than one "minimal functionally non-redundant set of application systems" there are also different solutions for the set covering problem. The simplest algorithm, known as "brute-force", checks all combinations of application systems for postulations (P1) and (P2). Of course this would need too much computing resources for realistic information systems with several tenths of application systems. Moreover, set covering is an NP-complete problem generally, which, roughly, means that the complexity of any algorithm will be in the order of an exponential function of N. In section 0 we will briefly sketch an algorithm which manages the situation of usual information systems quite well and we will report on the application of this algorithm in Leipzig in section 6. So we can assume here that we actually can find a solution for the problem. The solution is the set $\underline{USE}^{\min}$ of all vectors $\overrightarrow{USE_k}^{\min} := (use_{k,n}^{\min})_{n=1...N}$, for which (P1) and (P2) hold, is defined as

$$\underline{USE}^{\min} := \{\overrightarrow{USE_1}^{\min}, ..., \overrightarrow{USE_K}^{\min}\} \tag{11}$$

This corresponds with the set

$$\underline{AS}^{\min} := \{\underline{AS}_1^{\min}, ..., \underline{AS}_K^{\min}\} \tag{12}$$

of minimal functionally non-redundant sets of application systems $\underline{AS}_k^{\min}$. In the sense of the set covering problem we could say every $\underline{AS}_k^{\min}$ covers $\underline{EF}$. Because of (P2), all those sets $\underline{AS}_k^{\min}$ are of the same cardinality

$$M := \left|\underline{AS}_k^{\min}\right| \tag{13}$$

We can now define the key performance indicator *Functional Redundancy Rate (FRR)* as a measure for functional redundancy in an information system, which can be used for information management:

$$FRR := \frac{N - M}{N} \tag{14}$$

FRR can be interpreted as the percentage of application systems which could be removed from the information system without loss of functionality.

Example (part 3):
Since the given information system in part 1 of the example is quite small, we can immediately identify two minimal functionally non-redundant configurations:

$\underline{AS}_1^{\min} = \{1,2,4,5,6\}$ and $\underline{AS}_2^{\min} = \{1,2,4,6,7\}$ which correspond to the vectors $\overrightarrow{USE_1}^{\min} = (1,1,0,1,1,1,0,0,0)$ and $\overrightarrow{USE_2}^{\min} = (1,1,0,1,0,1,1,0,0)$.

For $\overrightarrow{USE_1}^{\min}$ as one of the two minimal solutions in our example holds: $SUP * \overrightarrow{USE_1}^{\min} = \left(\overrightarrow{ISUP_1}^{\min}\right)^T = (1,1,3,1,1,1,1)$ (see table 3).

Table 3. *Vector* $\overrightarrow{ISUP_1}^{\min}$

p	**ef$_p$**	**isup$_p$**
1	A	1
2	B	1
3	C	3
4	D	1
5	E	1
6	F	1
7	G	1

Thus (P1) holds for $\overrightarrow{ISUP_1}^{\min}$. In the same way (P1) can be shown to hold for $\overrightarrow{ISUP_2}^{\min}$ as well.

FRR is only dependent on *N* and on *M*, being the number of application components and the cardinality of all minimal functionally non-redundant sets of application systems, respectively. With *N*=9 and *M*=5, we get:

$$FRR = \frac{9-5}{9} = 0,44 \qquad (15)$$

Hence 44% of the application systems in our example could be removed.

4 Using the Functional Redundancy Rate and Minimal Non-redundant Sets of Application Systems to Support Information Management

This approach may be supportive for information management in different ways:

4.1 Benchmarking Information Systems

The Functional Redundancy Rate *FRR* may be used as a quality indicator, which supports benchmarking of information systems. Since besides the set of application systems the set of enterprise functions is one of the two input variables of *FRR*, it is obvious, that the structure (especially the granularity of the functions modelled) as well as the cardinality of this set will influence the result. Thus *FRR* depends on the individual way of modelling the enterprise functions in an institution and *FRR*s of different information systems may be incomparable. Moreover the *FRR*s derived by different models of different modellers of the same information system may differ as well. This problem can be overcome by using the same set of enterprise functions for models of those information systems which shall be benchmarked and compared. An appropriate set of enterprise functions for hospitals has recently been published as a reference model in [11]. Using this as a basis for *FRR* calculation can make information systems comparable with respect to their *FRR*. But of course complete modelling is needed anyway.

Example (part 4)
The Functional Redundancy Rate of 44%, which has been calculated in part 3 of the example, indicates that according to the model 44% of the application systems in this information system – 4 out of 9 – are superfluous. This is an indicator, that – given the model is sound and complete – information management in the respective hospital may not have been performed very systematically.

4.2 Reducing Operational Costs

Even if a particular application system cannot be shut down and omitted, it may cause unnecessary operational costs. Let $\underline{AK}_k^{\min}$ be a minimal set of used application systems and $isup_p$ the related individual degree of support by application systems in $\underline{AK}_k^{\min}$ for every enterprise function $ef_p \in \underline{EF}$. If for some *p* holds $isup_p^{\min} > 1$, this indicates, that users can use different application systems as support for the enterprise function ef_p Information managers should check, whether this option is really favored; since it may cause additional costs e.g. for customizing the different application systems the same way, providing catalogues of terms and diagnoses redundantly, additional training courses, and so on.

Example (part 5)
As can be seen in the vector $\overrightarrow{ISUP_1}^{\min}$ in part 3 of the example, users having to perform "administrative admission (outpatients)" (function *C*) have the option to choose between 3 application systems. Information management should check, whether it is appropriate to allow employees to choose between application systems *1* "CareMgmtSys", *2* "PatientAdministrationSystem(ADT)", and *4* "Departmental-SystemRadiotherapy", if they have to admit outpatients; because this option causes additional expenses e.g. for training. Information management could decide that only the "PatientAdministrationSystem(ADT)" has to be used for the admission of

outpatients and could block the respective modules of the “CareMgmtSys” and “DepartmentalSystemRadiotherapy”.

4.3 Shut Down of Superfluous Application Systems

Realizing the *FRR* of the information system the responsible chief information officer (CIO) will ask what application systems actually are superfluous and can be omitted. Using the concepts introduced before we can calculate the subset of those application systems, which are superfluous and can be omitted anyway. Other way round those application systems can be found, which by no means should be deleted. The calculation of the latter can simply be based on the matrix *SUP* (see formula (3)). An application system as_n cannot be deleted, exactly if there is an enterprise function ef_p such that as_n is the only application system supporting this function. Let us collect these application systems in the set:

$$\underline{AS}^{+} := \left\{ as_n \in \underline{AS} \mid \left(\exists ef_p \in \underline{EF} : (sup_{p,n} = 1) \wedge \left(\forall m \neq n : sup_{p,m} = 0 \right) \right) \right\} \qquad (16)$$

The calculation of superfluous application systems is more difficult. Of course all those application systems supporting no enterprise function can be omitted. Furthermore already matrix *SUP* provides valuable information concerning possible replacement of one application system by a different one. If we define two application systems functionally equivalent if they support the same set of enterprise functions, *SUP* can be used to determine those application systems which are mutually equivalent.

$$equal(as_x, as_y) = true \Leftrightarrow \forall p : sup_{p,x} = sup_{p,y} \qquad (17)$$

Doing so every application system could be replaced by one of its equivalents. Thus first decisions can be made, what application systems should be shut down. But there may be more superfluous application systems, which can be found by using the set $\underline{AS}^{\min} := \{\underline{AS}_1^{\min}, \dots, \underline{AS}_K^{\min}\}$ as defined in (12) resp. calculated before. Let us define the set

$$\underline{AS}^{-} := \bigcap_{k=1}^{K} \left(\underline{AS} \setminus \underline{AS}_k^{\min} \right) \qquad (18)$$

of those application systems, which have been found as *not* needed in *all* minimal sets $\underline{AS}_k^{\min}$. Thus the application systems in $\underline{AS}^{-}$ can be omitted anyway.

$$\underline{AS}^{?} := (\underline{AS} \setminus \underline{AS}^{-} \setminus \underline{AS}^{+}) \qquad (19)$$

Finally the set $\underline{AS}^{?}$ in (19) contains application systems which are not clearly marked as needed or not. But we can use the equivalence relation mentioned before to group the members of $\underline{AS}^{?}$ into equivalence classes. Based on this we have to decide for every equivalence class, what member of this class should be used; the rest of the class can be omitted.

Example (part 6)

Using *SUP* of part 1 of the example immediately results in $\underline{AS}^{+} := \{1,2,4,6\}$; these application systems are marked with ⊕ in figure 1 and must not be omitted. As stated in part 3 of the example, $\underline{AS}^{\min} = \{\{1,2,3,5,6\},\{1,2,4,6,7\}\}$. Thus $\underline{AS}^{-} = \{3,8,9\}$, which means, that the application systems marked with ∅ in figure 1 should be omitted anyway. The application systems marked with ⑨ in figure 1 belong to the set $\underline{AS}^{?} = \{5,7\}$. Since both support the same set of functions, they belong to the same equivalence class and on of them can be selected to support function E. Given *Clinical-PathwaySys* is a personal favourite of the hospital's medical director the CIO maybe decides for *ClinicalPathwaySys* and consequently shuts down the *KnowledgeService*. Finally the CIO can reduce the information system according to figure 2:

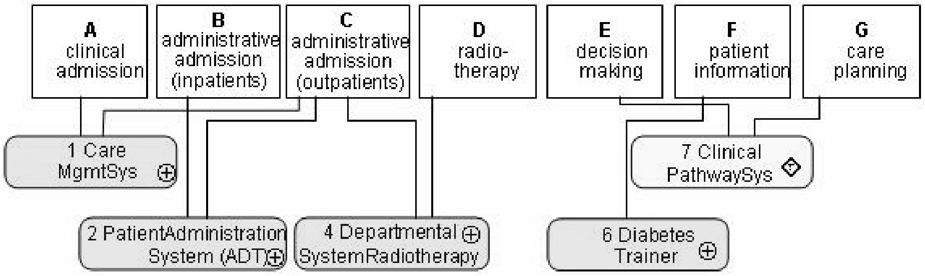

A clinical admission
B administrative admission (inpatients)
C administrative admission (outpatients)
D radio-therapy
E decision making
F patient information
G care planning
1 Care MgmtSys ⊕
7 Clinical PathwaySys
2 PatientAdministration System (ADT) ⊕
4 Departmental SystemRadiotherapy ⊕
6 Diabetes Trainer ⊕

Fig. 2. Reduced information system (caption see figure 1)

4.4 Exploiting Potentials of Application Systems and Reducing Heterogeneity

In section 2 we defined the matrix *SUP* for modeling the support of enterprise functions by application systems. But there may be cases, that particular application software products could support more enterprise functions than the actual implementation, i.e. the application system, does. If a modeler adjusts the matrix *SUP* in a way, that it maps an enterprise function not only to application systems, which actually support this enterprise function, but also to those application systems, which *could* do so, usually more potentially superfluous application systems may be identified.

Example (part 7)

An analysis of the application software product, underlying application system "CareMgmtSys", may turn out that by a proper installation this application system could also support function "patient information". In this case, the "DiabetesTrainer" could be omitted, too.

5 An Algorithm for Calculating Minimal Non-redundant Sets of Application Systems

Our approach is mainly based on the set $\underline{USE}^{\min} := \{USE_1^{\min},...,USE_K^{\min}\}$ of minimal vectors defining minimal non-redundant sets of application systems. But up to now we did not elucidate how to compute this set. As mentioned before, there are algorithms presented in literature, to solve the set covering problem. Far from starting a new discussion on optimal solutions for set covering problems in general we want to show the feasibility, i.e. the computability of the set $\underline{USE}^{\min}$ in real settings within acceptable time. According to our experiences there may be hundreds of application systems and enterprise functions in those settings. Applying set covering solving algorithms immediately would result in inacceptable computing efforts. But we have made also the following experiences:

E1. Most of the functions will be supported by exactly one application system. The corresponding application systems are members of $\underline{AS}^{+}$.

E2. Due to incomplete models there will be more or less application systems supporting none of the enterprise functions: $\underline{AS}^{0}$.

E3. There will be more or less application systems supporting only enterprise functions which are already supported by one of the application systems in $\underline{AS}^{+}$: $\underline{AS}^{-}$.

Thus we can reduce the set of application systems to $\underline{AS}^{reduced} := \left(\underline{AS} \setminus \underline{AS}^{+} \setminus \underline{AS}^{0} \setminus \underline{AS}^{-}\right)$. This set can be further reduced by using the equivalence relation (17) introduced in 0 and calculating the respective equivalence classes. Based on this we collect one (arbitrary) element from each class into the set $\underline{AS}^{equi}$. Now we can use $\underline{AS}^{equi}$ in place of $\underline{AS}$ to solve the set covering problem by one of the algorithms well known in literature delivering $\underline{AS}^{equi\min} := \{\underline{AS}_1^{equi\min}, \ldots, \underline{AS}_L^{equi\min}\}$ [10]. Based on this calculation and according to (17) we can calculate

$$\underline{AS}^{equi-} := \bigcap_{l=1}^{L} \left(\underline{AS}^{equi} \setminus \underline{AS}_l^{equi\min}\right) \tag{20}$$

If we take into account that every member of $\underline{AS}^{equi-}$ in fact is a place holder for an equivalence class, we can also derive $\underline{AS}^{?}$ as defined in 0.

6 Using the Functional Redundancy Rate and Minimal Non-redundant Sets of Application Systems at Leipzig University Medical Center

We implemented an algorithm in JAVA solving this set covering problem, which first reduces the set of application systems to be examined to $\underline{AS}^{equi}$ as described. $\underline{AS}^{equi}$ is explored using decision trees and a backtracking algorithm [12]. Using this algorithm we explored the 3LGM² model of the information system of Leipzig University Medical Center [13]. See table 3 for the results of the assessment.

Table 4. Analysis of the 3LGM² model of the information system of Leipzig University Medical Center

Functional Redundancy Rate (FRR)	25%
Number of application systems (N=$\lvert \underline{AS} \rvert$)	123
Number of application systems exclusively supporting functions ($\underline{AS}^+$)	86
Number M of needed application systems	92
Number of application systems supporting only functions which are already supported by one of the application systems in $\underline{AS}^+$ ($\underline{AS}^-$)	25
Number of equivalence classes	6
Number of application systems supporting no function ($\underline{AS}^0$)	0
Number of application systems to be examined by set covering solving algorithms	5
No. of redundant application systems found by set covering solving algorithms	4
Computing time (on a usual PC):	< 1 sec

The resulting *FRR* 25%, taken for itself, indicates that a quarter of the application systems could be removed without loss of functionality. The sets $\underline{AS}^?, \underline{AS}^-, \underline{AS}^+$ uncovered some interesting aspects regarding the model contents, e.g.:

- Since we modeled not only application systems of Leipzig University Medical Center but also of some hospitals in the neighbourhood, the algorithm suggested to omit the ADT-systems of these hospitals because the ADT-System of Leipzig University Medical Center would cover the functionality sufficiently.
- Two application systems supporting classification of diagnoses and procedures have been found as being superfluous. They can be omitted since a new system has been introduced some time before.

As shown in table 3 we have had quite small computing time; this is due to only 5 application systems to be examined by time-consuming algorithms.

7 Discussion

We have introduced the *Functional Redundancy Rate (FRR)* as a new key performance indicator for information systems. Even if redundancy of functions has been discussed in medical informatics in the context of functional integrity (e.g. [6, 14]) we could not find any formal and quantitative approach for computing a related key performance indicator before. Moreover it was surprising, that the set covering problem being discussed since many years turned out to be such a well suited formal description of the problem.

But the *FRR* and its use depends strongly on the solution of an NP-complete problem. Because of the NP-hardness, there is no way but to accept a possibly high running time of the algorithm. But this might not be a problem since a calculation time of a weekend or two would be acceptable for gaining a saving of several thousands of Euro. We proposed an algorithm to better manage the situation. Of course we could not proof the computability of *FRR* in all settings using our algorithm. But we could show its computability in a realistic setting giving reason to assume its usefulness in similar settings as well.

Based on the algorithm the possible questions of information managers cited in the introduction can be answered. The sets $\underline{AS}^{+}$ and $\underline{AS}^{-}$ deliver the application systems being crucial respectively being obsolete. Additionally the *Functional Redundancy Rate (FRR)* supports benchmarking between different information systems.

Besides the complexity of the underlying set covering problem there is the additional problem of collecting all enterprise functions, all application systems and all their relationships for the calculation of *FRR*. Of course these efforts don't pay for only calculating the *FRR*. But if information management has a thorough description of the information system at its disposal, perhaps by having used the 3LGM² tool [15] the calculation of *FRR* does not need any further efforts.

The *FRR* for functional redundancy is only one key performance indicator for quality of information systems. Especially data redundancy is one more extremely relevant problem. Future research has to examine this and its relationships with functional redundancy as well and hopefully can result in considerable steps towards a sound and complete theory of quality of information systems. Dealing with quality criteria like functional or data redundancy makes evident, that a distinct ontological basis is needed independently from modelling approaches used. We need a common, unified ontology for describing information systems – not only in health care. We consider 3LGM² to be a proposal for first steps in this direction.

Acknowledgements

Thanks to Ernst Schuster. He helped us to find the proper formulation for our optimization problem. Parts of work have been supported by grants of the Deutsche Forschungsgemeinschaft (DFG).

References

1. Kuhn, K.A., Giuse, D.A., Lapao, L., Wurst, S.H.: Expanding the scope of health information systems - from hospitals to regional networks, to national infrastructures, and beyond. Methods Inf. Med. 46(4), 500–502 (2007)
2. Lorenzi, N.M., Riley, R.T.: Managing technological change: organizational aspects of health informatics. Springer, New York (2004)
3. Haux, R.: Individualization, globalization and health-about sustainable information technologies and the aim of medical informatics. Int. J. Med. Inform. 75, 795–808 (2006)
4. Ammenwerth, E., Aarts, J., Berghold, A., Beuscart-Zephir, M., Brender, J., Burkle, T., et al.: Declaration of Innsbruck. Results from the European Science Foundation Sponsored Workshop on Systematic Evaluation of Health Information Systems (HIS-EVAL). IMIA Yearbook of Medical Informatics 2006. Methods Inf. Med. 45(suppl. 1), 121–123 (2006)
5. Talmon, J.: Evaluation and implementation: A call for action. IMIA Yearbook of Medical Informatics 2006. Methods Inf. Med. 45(suppl. 1), 16–19 (2006)
6. Haux, R., Winter, A., Ammenwerth, E., Brigl, B.: Strategic Information Management in Hospitals. Springer, New York (2004)

7. Cimino, J.J., Zhu, X.: The Practical Impact of Ontologies on Biomedical Informatics. IMIA Yearbook of Medical Informatics 2006. Methods Inf. Med. 45(suppl. 1), 124–135 (2006)
8. Winter, A.F., Ammenwerth, E., Bott, O.J., Brigl, B., Buchauer, A., Gräber, S., Grant, A., Häber, A., Hasselbring, W., Haux, R., Heinrich, A., Janssen, H., Kock, I., Penger, O.-S., Prokosch, H.-U., Terstappen, A., Winter, A.: Strategic Information Management Plans: The Basis for systematic Information Management in Hospitals. International Journal of Medical Informatics 64(2-3), 99–109 (2001)
9. Winter, A., Brigl, B., Wendt, T.: Modeling Hospital Information Systems (Part 1): The Revised Three-Layer Graph-Based Meta Model 3LGM2. Methods Inf. Med. 42(5), 544–551 (2003)
10. Karp, R.M.: Reducibility Among Combinatorial Problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum, New York (1972)
11. Hübner-Bloder, G., Ammenwerth, E., Brigl, B., Winter, A.: Specification of a reference model for the domain layer of a hospital information system. Stud. Health Technol. Inform. 116, 497–502 (2005)
12. Cormen, T.H., Leiserson, C., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
13. Winter, A., Brigl, B., Funkat, G., Häber, A., Heller, O., Wendt, T.: 3LGM^2-Modeling to Support Management of Health Information Systems. International Journal of Medical Informatics 76(2-3), 145–150 (2007)
14. van Bemmel, J.H. (ed.): Handbook of Medical Informatics. Springer, Heidelberg (1997)
15. Wendt, T., Häber, A., Brigl, B., Winter, A.: Modeling Hospital Information Systems (Part 2): Using the 3LGM2 Tool for Modeling Patient Record Management. Methods Inf. Med. 43(3), 256–267 (2004)

The First International Workshop on Non-Functional System Properties in Domain Specific Modeling Languages (NFPinDSML2008)

Marko Bošković[1,⋆], Dragan Gašević[2], Claus Pahl[3], and Bernhard Schätz[4]

[1] TrustSoft Graduate School, University of Oldenburg, Germany
[2] School of Computing and Information Systems, Athabasca University, Canada
[3] School of Computing, Dublin City University, Ireland
[4] Institute for Informatics, Technical University Munich, Germany

Abstract. This workshop brought together researchers and practitioners from communities dedicated to non-functional properties of software systems and researches from language engineering to study the principles of integration of various non-functional system properties and language engineering in order to further expand principles of reasoning about non-functional properties of software systems in Domain Specific Modeling Languages, and model-driven engineering in general

1 Motivation

For the engineering of systems of a particular domain, Domain Specific Modeling Languages (DSMLs) are becoming a common-place in software and system engineering. While DSMLs are mostly dedicated to functional requirements, often they do not address non-functional system properties (e.g. availability, reliability, security, performance, timeliness, efficiency). Non-functional system properties (NFSP) are recognized as at least as important as functional properties and have to be addressed during the design of systems. As till now, the study of engineering DSMLs and analysis of NFSP lack common principles. For this reason the workshop gathered a forum discussing issues of integration of NFSP estimation and evaluation in the context of software system engineering with DSMLs.

2 Workshop Format

The workshop consisted of four sessions where position (short), and full technical (long) contribution papers were presented. Short papers discussed thought-provoking not yet fully developed and evaluated ideas, and long papers described fully developed new research results.

The first session was the presentation of invited speaker Prof. Dr. Dorina Petriu, from Carleton University, Ottawa, Canada. She presented the emerging

⋆ This work is supported by the German Research Foundation (DFG), grant GRK 1076/1.

results of her research group entitled "Performance Analysis of Aspect-Oriented UML Models".

The second session started with Richard Paige's long paper presentation entitled "Automated Safety Analysis for Domain-Specific Modeling Languages". He presented an automated safety analysis technique, for automatically calculating the failure behaviour of an entire system from the failure behaviours of its components.

He was followed by a long paper presentation of the paper entitled "An Extensible Services Orchestration Framework through Concern Composition", and presented by Gabriel Perdraza-Ferreira. "An Extensible Services Orchestration Framework through Concern Composition", presented by Gabriel Perdraza-Ferreira. Next, Moran Kupfer presented the short paper "Understanding and Representing Deployment Requirements for Achieving Non-Functional System Properties". The authors of the paper proposed a research direction towards developing an approach for reasoning about deployment decisions. Finally, Andreas Petter presented a short paper, "Modeling Usability in Model-Transformations". In their work they identified commonly needed features of transformation languages for implementation of transformations for usable user interfaces.

The third session was opened by a long paper presentation of Jon Whittle on detection of interacting aspects, entitled "Towards Semantic-Based Aspect Interaction Detection" authored by Gunter Mussbacher, himself, and Daniel Amyot. This presentation was followed by a long paper presentation of Christiano Braga "From Access Control Policies to an Aspect-based Infrastructure: A Metamodel-based Approach" on a validated process of code generation for role based access controll policies. The session was concluded with Jan Jürjens's position paper presentation, "Challenges for the Model-based Development of Distributed Real Time Systems" and authored by Michael Giddings, himself, and Pat Allen.

The four session was dedicated to discussion of open issues in DSML engineering and non-functional system properties estimation and evaluation. Some of the identified topics for future work are: identification of NFSP for particular domains, processes for decision making with multiple NFSP, propagation of values NFSP to requirements models from design models, transformations and code generation for optimization of NFSP, methodologies for identification of causes of problems with NFSPs, and generalities of language engineering and possibilities of existing analytical and simulation models integration.

3 Workshop Outline

The workshop lasted one day and had 25-30 participants. Four technical contributions and three position papers were presented. All of them are published as CEUR on line proceedings Vol-394. Furthermore, it was followed by a special issue of the Journal on Software and Systems Modeling for which papers were solicited by an open call. Finally, the plans for future of the workshop are to growing into a network of excellence dealing with issues of NFPinDSML.

FPTC: Automated Safety Analysis for Domain-Specific Languages

Richard F. Paige, Louis M. Rose, Xiaocheng Ge, Dimitrios S. Kolovos, and Phillip J. Brooke

Department of Computer Science, University of York, UK
School of Computing, University of Teesside, UK
{paige,louis,xchge,dkolovos}@cs.york.ac.uk, pjb@scm.tees.ac.uk

Abstract. Critical systems must be shown to be acceptably safe to deploy and use in their environment. The size, scale, heterogeneity, and distributed nature of these increasingly complex systems makes them difficult to verify and analyse. Additionally, domain experts use a variety of languages to model and build their systems. We present an automated safety analysis technique, Fault Propagation and Transformation Analysis, and explain how it can be used for automatically calculating the *failure behaviour* of an entire system from the failure behaviours of its components. We outline an implementation of the technique in the Epsilon model management platform, allowing it to be used with state-of-the-art model management languages and tools, and making it applicable to a variety of different domain-specific modelling languages.

1 Introduction

Complex systems exhibit emergent properties as a result of composing heterogeneous components. These components may be distributed, and may also have substantial performance, timing, safety, and security requirements. The scale and complexity of these systems make it difficult to apply general-purpose verification and validation technology – such as model checkers, simulators, and theorem provers – to obtain the guarantees of acceptable behaviour that are required. Obtaining guarantees is particularly important for safety critical systems, which normally must be certified as acceptably safe according to relevant standards before they are deployed in the field.

Safety analysis for complex systems is an open field of research. For high-integrity real-time systems (HIRTS), *automated* safety analysis can help to achieve the substantial requirements for reliability and safety necessary for these kinds of systems to achieve certification. Safety analysis techniques are novel when contrasted with traditional software analyses, which tend to emphasise determining a product's correctness – e.g., through proof, model checking, simulation, or abstract interpretation. For HIRTS, it is of critical interest to know how a system behaves in the presence of failure, regardless of whether that failure is in the environment, or due to internal software or hardware error. Given an understanding of a system's behaviour in the presence of failure, methods to mitigate potential hazards can be determined and engineered.

Manual safety analysis is notoriously expensive for all systems; anything that can be done to help to automate the process of understanding system behaviour in the presence of failures will be of benefit to industry. Moreover, safety analysis is not in general compositional – even small changes to components (and their corresponding failure behaviour) generally means that the whole safety analysis has to be performed again for the entire system. Finally, engineers of safety critical systems often use a variety of general purpose and domain-specific languages (DSLs), including profiles of UML, Matlab, Simulink, Stateflow, AADL, SysML, and MASCOT [14]; safety analysis that is applicable to all these languages (and others) will be of substantial value.

This paper presents a fully automated and compositional safety analysis technique applicable to domain-specific languages. The technique, *fault propagation and transformation analysis*, is outlined and an implementation in the Epsilon model management toolset [11], is described in detail. The value of having an implementation in a state-of-the-art and standards compliant toolset like Epsilon, built atop of Eclipse, is also explained, particularly for supporting analysis on domain-specific and heterogeneous models.

2 Background and Related Work

2.1 Components and Failures

Failure and safety analysis is generally applied to component or architectural models of safety critical systems. In these models (which may be represented using one of a number of different languages), a component is a building block of a system, and may be represented in hardware or software. A component has input ports and/or output ports, and transfers inputs to outputs. Components may exhibit expected behaviour (e.g., according to a specification such as a pre- and postcondition), but may also exhibit *failures.* A failure is any behaviour of a component or system which deviates from specified behaviour [5]. Failures arise, can be propagated, and can also be transformed in a system, e.g., as a result of an accident or incorrect implementation. In order to determine the failure behaviour of a system, it is necessary to be able to understand, and model, the failure behaviours of the system's components.

There has been previous work on modelling and understanding the failure behaviour of systems. Traditional safety engineering techniques include Failure Modes and Effects Analysis (which is a manual process) [7], HAZOPs guidewords-based analyses, Fault Tree Analysis, and finite state analysis [6]. Fenelon et al introduced FPTN [4], a notation for explicitly representing the failures behaviour of components, and integrated FPTN with a typical safety engineering process. However, FPTN possessed no tool support. Overall, few of these approaches have been integrated with Model-Driven Engineering standards and tools. An exception is work on integrating Fault Tree Analysis with UML, e.g., as carried out by Jürjens [8]; this integration was specifically for UML, and did not support automated compositional reasoning about failures.

Wallace [15] proposed using a model of HIRTS system architecture as the basis for safety analysis. The failure behaviours of the components are determined and modelled when analysing the system. The connections between units are communication protocols. Because a communication protocol also has its own potential failure behaviour, the protocols in the model must be treated identically to the components of the system – i.e., their failures are also modelled.

In [15], a component can introduce failures (e.g., because of an exception or crash), or may propagate failures (e.g., data that is erroneous when it arrives at a component remains erroneous when it leaves the component), or transforms a failure into a different kind of failure (e.g., data that arrives late may thereafter arrive early). Furthermore, a component may correct or mask failures that it receives. Thus, when a component receives as input a particular kind of failure, it generates one of the following responses.

$$output = \begin{cases} normal \\ same\ failure \\ different\ failure \end{cases}$$

To support automated failure analysis, we must be able to connect models of component failure behaviour to a system model. This can be done by representing the behaviour of architectural components – such as hardware, wires, and network connections – using failure models. In our implementation in Epsilon, we do this by effecting a model transformation (though it is not a traditional *mapping* transformation in the classification of Czarnecki [2]).

2.2 Automating Failure Analysis

Assume that we have a component-and-connector model of a system, e.g., in a DSL. The components in the system can be individually analysed – in isolation, from the rest of the system – for their failure behaviour in response to potential failure stimuli. This behaviour should be determined by domain and safety experts, and should consider all possible failures on input. The analysis can follow the conventional HAZOP/SHARD [13] identification of *types* of failures through a set of *guidewords*, such as: value failures (e.g., data is stale); timing failures (e.g., data is arriving later); and sequence failures (e.g., omission).

Following [15], we capture the failure responses of a component to its input in a simple pattern-based modelling language. Using $\star$ to indicate normal (no failure) behaviour, the following four expressions denote example source, sink, propagation, and transformation behaviours for a trivial single-input single-output component (the generalisation to multiple inputs and multiple outputs is straightforward, and is illustrated in Section 4).

$$\begin{array}{rll} * \rightarrow & \text{late} & \textit{(failure source)} \\ \text{early} \rightarrow & * & \textit{(failure sink)} \\ \text{omission} \rightarrow & \text{omission} & \textit{(failure propagation)} \\ \text{late} \rightarrow & \text{value} & \textit{(failure transformation)} \end{array}$$

The first line says that any input leads to late output, whereas the second says that early input (data arriving before its time) leads to no error, i.e., the component sinks all errors. The third line says that an omission failure is propagated by the component. The most complicated failure behaviour is generally transformational – i.e., the last line above, where a late input leads to a value error — the wrong value being output.

A typical component will have its failure behaviour modelled by a number of patterns of this form, and the cumulative effect is its overall *behaviour*. [15] calls this the *FPTC behaviour* of a component.

To represent the system as a whole, every element of the architectural model – both components and connectors – is assigned FPTC behaviour. Given this, we can automatically calculate the failure behaviour of a whole system as follows (see [15] for a formal definition). Each model element that represents a relationship is annotated with sets of *tokens* (e.g., late, early, value), which represent all possible failures that can be propagated by this dependency. In other words, we are informally treating the architectural model as a token-passing network. As a result of this annotation, we can calculate the failure behaviour of the system by calculating the *maximal* token sets on all dependencies in the model. This turns out to be a fixpoint calculation (presented formally in [15]). Informally, the calculation works as follows. Starting with the singleton set containing the *no failure* ($\star$) token as a label on every dependency, the FPTC behaviour at every component model element is 'run', using the token sets on input dependencies as the inputs to the FPTC behaviours. The output failure tokens of each component are accumulated on the outgoing dependencies, and the system continues to run until a fixed point is reached, i.e., the token sets no longer change.

The calculation must terminate, because the set of failure types must be finite. [15] also shows that the calculation produces the same result no matter in what order the relationships are analysed.

3 Implementation in Epsilon

We have implemented the failure analysis in the Epsilon model management platform[1], under Eclipse. By using Eclipse, we can exploit its mechanisms for metamodelling, modelling, and extension, as well as its substantial tool support via plug-ins. Epsilon itself provides a *model management* framework, via a suite of integrated languages. Epsilon provides a base language – the Epsilon Object Language (EOL) [12] – which supports basic model manipulation, e.g., traversal of models, querying models, modifying models. EOL has many similarities to the OMG-standard Object Constraint Language (OCL), but is fully executable and metamodel-independent; thus, EOL (and Epsilon) can be used to manage models from any language. By implementing the failure analysis approach within Epsilon, we thus immediately obtain independence from UML-based languages, but also technology independence, because EOL can be used to manage models from different technologies such as EMF, MDR/MOF, Z models, and XML.

[1] www.eclipse.org/gmt/epsilon

The Epsilon platform includes other model management languages that have been built on, and thus inherit from, EOL. These include a model-to-model transformation language, a model merging language, a model-to-text transformation language, a validation language, and a refactoring language, amongst others. Further details can be found in [11].

The FPTC analysis has been implemented and encoded directly in EOL, based on a lightweight and reasonably generic metamodel for architectural modelling; the metamodel is shown in Fig. 1. EOL was used because the model management task to be completed – calculating a fixpoint on a model – is iterative, and EOL is the only language in the platform providing iterative constructs. The metamodel that we use as the basis of the FPTC calculations is intended to be generic so that it can (a) provide sufficient infrastructure for the FPTC calculations; and (b) make it reasonably straightforward to use as the target of model-to-model transformations from other architectural modelling DSLs, such as UML 2.x, SysML, and AADL. We have implemented several simple transformations for such source languages.

In the architectural language of Fig. 1, systems are made up of blocks (which represent both components and connectors). The system overall, and individual blocks, have fault behaviour, represented as expressions. Expressions are made up of a number of tuples (which correspond to the patterns we discussed earlier).

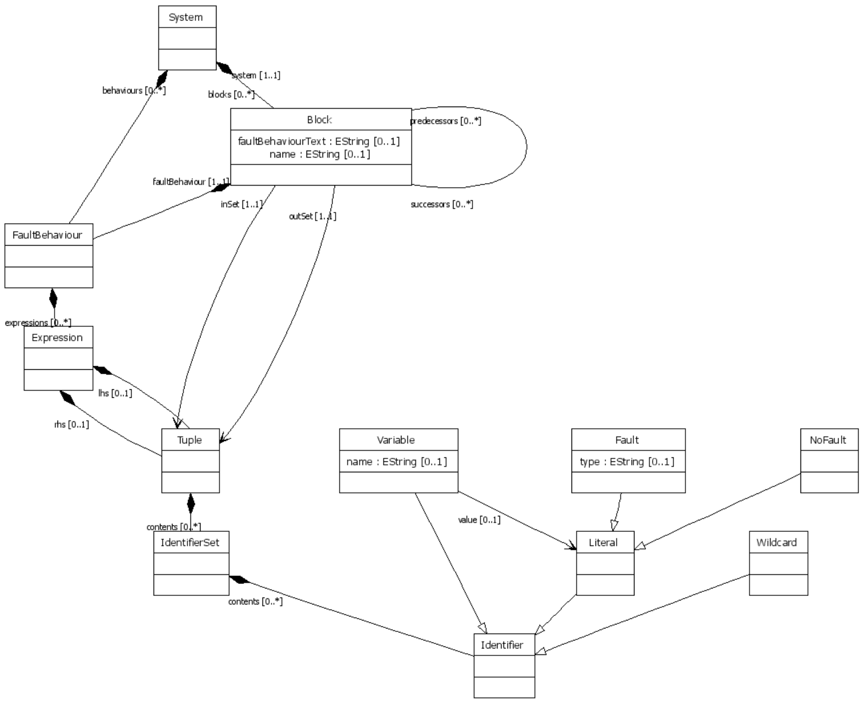


Fig. 1. Example metamodel for architectural modelling

These tuples include sets of identifiers, where an identifier can be a wildcard (i.e., no-fault behaviour), a literal (i.e., a domain-specific kind of fault), or a variable.

The actual implementation, written in EOL, encodes the algorithms described earlier. A certain amount of simple EOL infrastructure needs to be provided (e.g., to record the behaviour of blocks, variables, and literals, and to reset these behaviours across different runs of the analysis). The remainder of the implementation is more complex, and can be subdivided into four main parts (not including any visual representation of the output of the analysis, nor how individual FPTC behaviours are expressed – we discuss this afterwards):

– the pattern matching, e.g., to match failure behaviours with inputs; this is written as a *model comparison* operation in EOL. We could equally do this in the Epsilon Comparison Language (ECL) but because the pattern matching that we need to do is straightforward and not rule-based – and we need to use the results of the matching in further EOL programs – we encode the comparison directly in EOL. This is one of the benefits of having an executable base language in Epsilon that is also computationally complete.
– the propagation behaviour, i.e., what happens when a component or connector propagates failure behaviour to its environment. As this is an algorithmic calculation, we implement this with EOL.
– the transformation behaviour, i.e., what happens when a component or connector generates new failure behaviour to its environment, based on specific input behaviour. This could be implemented using the Epsilon Transformation Language (ETL) [9] or EOL. We chose the latter for reasons similar to the model comparison phase: the transformation we need to carry out is not a mapping, and is predominantly algorithmic instead of rule-based. As such, EOL was a better fit for this transformation problem versus ETL.
– the overall system analysis, which is a fixpoint calculation over the system model.

The matching behaviour in EOL is described in Listing 1.1. This implements a pattern matching on blocks and sets of identifiers. The pattern matching is implemented as a set of overloaded operations, called *matches*; one *matches* operation is defined for each type of model element that can be matched, e.g., blocks, faults, variables, identifier sets, etc. Effectively, each operation simple compares an input (consisting of failure behaviour) against the behaviour of a model element and returns true or false. We show three examples: for matching sets of identifiers, for matching faults, and for matching no-fault behaviour; other match operations are direct transliterations of the ones we show.

Listing 1.1. EOL Pattern Matching

```
operation IdentifierSet matches(inSet : IdentifierSet) : Boolean {

   for (identifier in self.contents) {
      for (inSetIdentifier in inSet.contents) {
         if (identifier.matches(inSetIdentifier)) {
            return true;
         }
      }
   }
   return false;
}

operation NoFault matches(identifier : Identifier) : Boolean {
   if (identifier.isTypeOf(NoFault)) {
      return true;
   } else {
      return false;
   }
}

operation Fault matches(identifier : Identifier) : Boolean {
   if (identifier.isTypeOf(Fault)) {
      return identifier.type = self.type;
   } else {
      return false;
   }
}
```

Failure propagation behaviour is illustrated in Listing 1.2. This EOL program calculates output behaviour of a block from input behaviour. Effectively, when the *propagate* operation is applied to a specific block in an architectural model, it iterates through all successor blocks (i.e., all blocks that it is connected to). After obtaining the input token set for the current block, it simply propagates all input faults to the output block.

Listing 1.2. EOL propagation behaviour

```
operation Block propagate() {
  var index : Integer := 0;

  for (successor in self.successors) {
       var inSet : IdentifierSet;
       var outSet := self.outSet.contents.at(index);

       -- Retrieve the corresponding in-set for this out-set
       var pIndex : Integer := 0;
       for (predecessor in successor.predecessors) {
       if (predecessor = self) {
            inSet := successor.inSet.contents.at(pIndex);
         }
         pIndex := pIndex + 1;
       }

       -- Propagate identifiers from out-set to in-set
       for (identifier in outSet.contents) {
         inSet.contents.add(identifier.clone());
       }
       index := index + 1;
  }
}
```

The transformation behaviour is the most complicated part of the analysis. The EOL program implementing the transformation calculates new failure behaviour from input behaviour. It applies the *matches* operations presented earlier to match input failure behaviours against failure behaviours of the component. If there is a match on the left-hand side of a pattern, then the right-hand side failure behaviour is generated on the output of the block. The main part of the functionality is in operation *transform*, shown below.

Listing 1.3. EOL transformation behaviour

```
operation Block transform() : Boolean {
  -- Determine which expressions match
  var applicable : Sequence(Expression);
  for (exp in self.faultBehaviour.expressions) {
       if (exp.lhs.matches(self)) {
          applicable.add(exp);
   }
  }

  var result : Boolean := false;
  var selected : Expression;

  self.toString().println();

  if (applicable.size() > 0) {
    if (applicable.size() = 1) {
       selected := applicable.at(0);
    } else {
          selected := applicable.mostSpecific();
    }
      selected.toString().println();
    result := selected.applyTo(self);
    self.toString().println();
  }
  ''.println();

  return result;
}
```

There are a few subtleties to implementing the transformation behaviour. When matching input failures against failure behaviours of a component, there may be several matches; this is recorded in the EOL program via the variable *applicable*. This is an artefact of allowing wild-card specifications of behaviour (i.e., any fault is matched). To deal with this issue, we always select the most specific match; this is implemented in an EOL operation called *mostSpecific()*.

The second subtlety is in copying failure values to the output of a block. A component may have several failures on its outputs (and indeed, it may have many outputs), and we must be careful to record all of them in the output expressions for the block. This is handled in the EOL operation applyTo(), which applies a model of failure behaviour to a block's inputs. While we omit the details of applyTo(), it is a good example of a transformation that is not inherently a mapping, and which would be more concise expressed using an algorithmic specification. Another example of such a transformation was presented by Conmy [1], where the transformation was intended to generate large numbers of stable configurations of an adaptive system. These transformations are similar because both involve iterative processing of models, rather than rule-based processing.

Finally, the overall system analysis is encoded in Listing 1.4. The failure analysis is launched on the full system by the EOL run-time. The analysis first initialises all blocks with their failure behaviour, and then calculates the output sets on all blocks, until no output set changes, i.e., a fixpoint has been reached.

Listing 1.4. EOL failure analysis

```
System.allInstances().at(0).doFailureAnalysis();

operation System doFailureAnalysis() {
 for (block in Block.allInstances()) { block.initialise(); }

 var blocksChanged : Boolean := true;

 while (blocksChanged) {
  blocksChanged := false;

  -- Calculate out sets
  for (block in Block.allInstances()) {
      blocksChanged := block.transform() or blocksChanged;
  }

  -- Calculate new in sets
  for (block in Block.allInstances()) { block.resetInSet(); }
  for (block in Block.allInstances()) { block.propagate(); }
  ('=============================================').println();
 }
}
```

Epsilon is integrated with Eclipse GMF, so it is possible to create customised GMF editors and visualisations of the results of the FPTC analysis, and of the architectural models themselves. We present an example of this in the next section.

4 Example

In this section, we introduce a concrete example to illustrate the functionality provided by our FPTC implementation. Consider the architectural model shown in Figure 2. The model is written in a DSL for real-time systems. The depicted system comprises four software components, connected using three instances of a signalling communication protocol. This protocol uses a destructive (non-blocking) write, and a destructive (blocking) read.

We have used a simple GMF editor for creating this model. The model must now be transformed to include failure behaviour, so that we can perform the failure analysis. FPTC behaviours are most easily expressed in a textual format. In order to support this, we have integrated Epsilon with a model-generating parser specified using oAW's xText [3]. This allows the FPTC behaviour to be

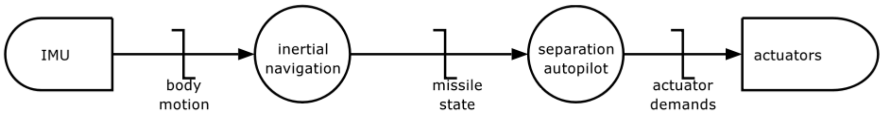


Fig. 2. Architectural model of the exemplar system

Table 1. Behavioural properties of components

Component	Behaviour
Inertial Navigation	v → v
Separation Autopilot	* → *stale_value* * → *detectable_value* v → v
Signal Comms Protocol	*early* → * *omission* → *late* *commission* → *value* v → v

easily expressed in a format similar to what was presented earlier. The FPTC behaviours of the individual components and connectors is listed in Table 1. These behaviours have been determined by domain experts knowledgeable about the individual components and connectors and their properties.

These experts have determined that the inertial navigation and separation autopilot components both propagate any faults that they receive. In addition the separation autopilot component acts as a source for *stale_value* and *detectable_value* faults. The signalling communication protocol exhibits a rather more complex failure behaviour, and comprises three non-trivial expressions. The first states that, as the protocol utilises a blocking read, should the supplier provide a value earlier than the receiver expects, no fault is produced. In the case where the communications protocol fails to relay a message (an omission), the receiver may block indefinitely, causing it to be delayed (encoded as a late fault). When the communications protocol duplicates a message sent from the supplier (a commission), the receiver may proceed with an incorrect value. Additionally, the protocol simply propagates all other categories of fault.

Having used our GMF-based editor to record the results of our behavioural analysis of the individual components, we can inject various different types of faults to potential sources of errors, and run the FPTC analysis to determine how the system would respond to these types of failures. For example, injecting an omission fault on the IMU component and executing our simulation toolchain yields the results depicted in Figure 3. By examining the faults produced by the actuator demands, it can be seen that the actuators may receive faults from the set {*, *stale_value*, *detectable_value*, *late*}.

Suppose the design is now changed such that a second IMU is introduced in order to provide two-lane redundancy. Additionally, instead of the signal, a pooling communications protocol is used between the IMU and inertial navigation components. Unlike the signalling protocol, the pooling protocol

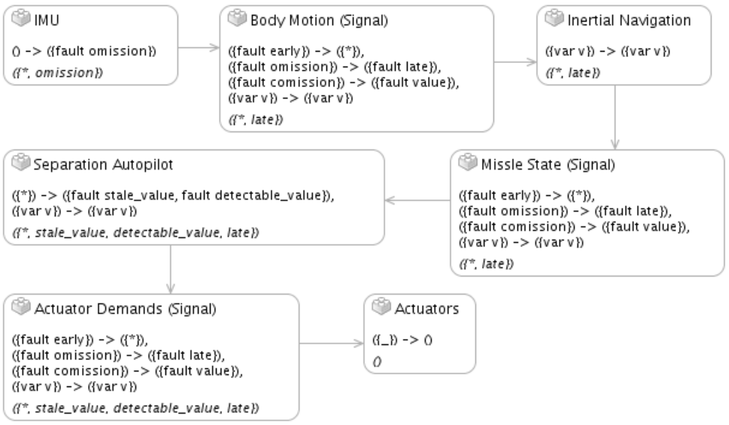

IMU
() -> ({fault omission})
({*, omission})
Body Motion (Signal)
({fault early}) -> ({*}),
({fault omission}) -> ({fault late}),
({fault comission}) -> ({fault value}),
({var v}) -> ({var v})
({*, late})
Inertial Navigation
({var v}) -> ({var v})
({*, late})
Separation Autopilot
({*}) -> ({fault stale_value, fault detectable_value}),
({var v}) -> ({var v})
({*, stale_value, detectable_value, late})
Missle State (Signal)
({fault early}) -> ({*}),
({fault omission}) -> ({fault late}),
({fault comission}) -> ({fault value}),
({var v}) -> ({var v})
({*, late})
Actuator Demands (Signal)
({fault early}) -> ({*}),
({fault omission}) -> ({fault late}),
({fault comission}) -> ({fault value}),
({var v}) -> ({var v})
({*, stale_value, detectable_value, late})
Actuators
({_}) -> ()
()

Fig. 3. Results of executing the simulation on the system. The faults produced by each component are shown in italics.

Table 2. Behavioural properties of the components in the simple system

Component	Behaviour
Pool Comms Protocol	*early* → *
	omission → *stale_value*
	late → *stale_value*
	commission → *
	v → v

provides a buffer from which the receiver may non-destructively read. This leads to the rather different failure behaviour shown in Table 2. Should the pooling protocol omit a message from receiver to supplier, the receiver may read a previous (stale) value from the buffer. Similarly, if the supplier is late sending a message to the receiver, the receiver may read a stale value from the buffer. Finally, if the protocol duplicates a message sent from supplier to receiver, the receiver proceeds as expected, due to the data being buffered by the protocol.

As can be seen in Figure 4, executing the simulation on the new model highlights that the set of faults propagated to the actuator from the actuator demands is {*, *stale_value*, *detectable_value*}. As such, we can conclude that the new model provides mitigation against faults with the categorisation *late*, whereas the original model does not.

This example illustrates the results that can be obtained by applying FPTC, and the lightweight nature of its analysis – we were able to change the architectural model to introduce different failure, and re-run the analysis to calculate the overall effect on the system. The ability to automatically and quickly analyse models makes the failure analysis technique particularly valuable for complex and critical systems development.

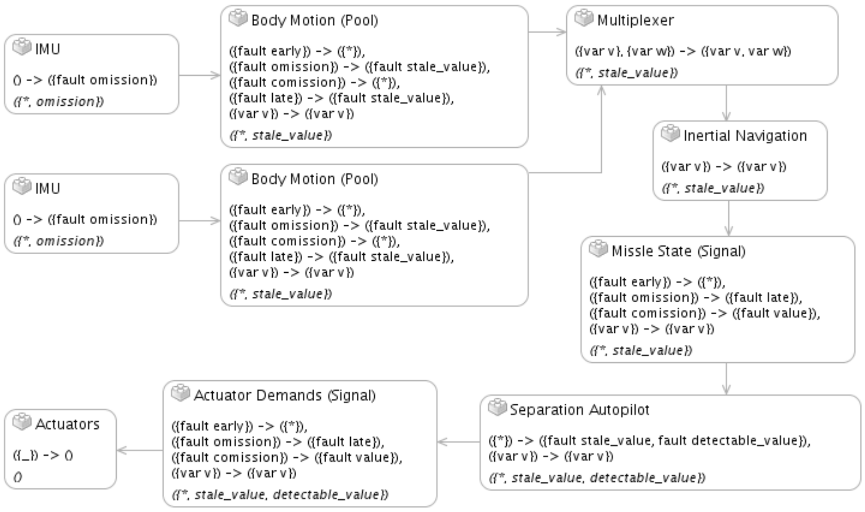

IMU
() -> ({fault omission})
({*, omission})
Body Motion (Pool)
({fault early}) -> ({*}),
({fault omission}) -> ({fault stale_value}),
({fault comission}) -> ({*}),
({fault late}) -> ({fault stale_value}),
({var v}) -> ({var v})
({*, stale_value})
IMU
() -> ({fault omission})
({*, omission})
Body Motion (Pool)
({fault early}) -> ({*}),
({fault omission}) -> ({fault stale_value}),
({fault comission}) -> ({*}),
({fault late}) -> ({fault stale_value}),
({var v}) -> ({var v})
({*, stale_value})
Multiplexer
({var v}, {var w}) -> ({var v, var w})
({*, stale_value})
Inertial Navigation
({var v}) -> ({var v})
({*, stale_value})
Missle State (Signal)
({fault early}) -> ({*}),
({fault omission}) -> ({fault late}),
({fault comission}) -> ({fault value}),
({var v}) -> ({var v})
({*, stale_value})
Separation Autopilot
({*}) -> ({fault stale_value, fault detectable_value}),
({var v}) -> ({var v})
({*, stale_value, detectable_value})
Actuator Demands (Signal)
({fault early}) -> ({*}),
({fault omission}) -> ({fault late}),
({fault comission}) -> ({fault value}),
({var v}) -> ({var v})
({*, stale_value, detectable_value})
Actuators
({_}) -> ()
()

Fig. 4. Results of executing the simulation on the modified system

5 Discussion and Conclusions

There are two additional, technical points regarding the FPTC implementation in Epsilon that are worth noting:

– Initial implementations of the failure analysis did not provide detailed error checking, e.g., to ensure that erroneous or inconsistent failure behaviours were not specified for components. For example, consider a component which was accidentally specified to deliver data both late and early; this should ideally be caught statically, before the FPTC calculation has been run. As part of the Eclipse/EOL implementation, we have exploited the availability of the Epsilon Validation Language (EVL) [10] for specifying well-formedness rules and constraints on the model. This helps us catch errors at an early stage. This consistency/constraint checking is also fully automated.
– We have provided support for constructing customised graphical interfaces by integrating Eclipse's GMF (Graphical Modelling Framework) with EOL. This allows the results of the failure analysis to be presented in ways suitable and appropriate for exploration by domain experts. This in itself is a novelty and provides functionality that is generally useful, not just for FPTC.

We have applied the FPTC analysis as part of work carried out in the Defense and Aerospace Research Partnership project at the University of York, in collaboration with BAE Systems, Rolls-Royce, QinetiQ, and the Ministry of Defense. The FPTC toolset has been applied to a number of case studies of very different models. The results demonstrated that the analytic technique was (a) scaleable;

(b) efficient; and (c) produced insightful and sometimes unexpected results. We are currently working on extensions to FPTC to support probabilistic analysis, i.e., where engineers can indicate the probability of particular types of failures occurring. This requires extension not only to the theory underpinning FPTC (specifically, the calculation of output token sets becomes much more complicated, since conditional probabilistic reasoning must be used), but also some extensions to Epsilon in order to efficiently support matrix calculations, which are an appropriate way to implement probabilistic analysis. As well, we are developing transformations for popular architectural modelling languages (such as AADL and SysML) into the analysis metamodel presented in this paper, so that FPTC can be used for these languages as well.

Acknowledgements. The work in this paper is partially supported by the European Commission via the MODELPLEX project, co-funded under the "Information Society Technologies" Sixth Framework Programme (2006-2009), and the EPSRC under the Large-Scale Complex IT Systems project, supported by research grant EP/F001096/1.

References

1. Conmy, P., Paige, R.: Challenges when using Model-Driven Architecture in the development of safety critical software. In: Proceedings of 4th Workshop on Model-Based Methodologies for Pervasive and Embedded Software. IEEE Computer Society Press, Los Alamitos (2007)
2. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal 45(3), 621–646 (2006)
3. Efftinge, S.: xText reference document (2007), `www.eclipse.org/gmt/oaw`
4. Fenelon, P., McDermid, J.A.: An integrated toolset for software safety analysis. The Journal of Systems and Software 21(3), 279–290 (1993)
5. Grunske, L.: Towards an integration of standard component-based safety evaluation techniques with saveCCM. In: Hofmeister, C., Crnković, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 199–213. Springer, Heidelberg (2006)
6. Heitmeyer, C.L., Kirby, J., Labaw, B.G., Archer, M., Bharadwaj, R.: Using abstraction and model checking to detect safety violations in requirements specifications. IEEE Trans. Software Eng. 24(11), 927–948 (1998)
7. IEC. Analysis techniques for system reliability: Procedures for failure mode and effect analysis. International Standard 812. IEC Geneva (1985)
8. Jürjens, J.: Model-based security engineering with UML. In: Aldini, A., Gorrieri, R., Martinelli, F. (eds.) FOSAD 2005. LNCS, vol. 3655, pp. 42–77. Springer, Heidelberg (2005)
9. Kolovos, D., Paige, R., Polack, F.: The Epsilon Transformation Language. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 46–60. Springer, Heidelberg (2008)
10. Kolovos, D., Paige, R., Polack, F.: On the evolution of OCL for capturing structural constraints in modelling languages. In: Rigorous Object-Oriented Methods. Springer, Heidelberg (2008)
11. Kolovos, D.S., Paige, R.F.: Epsilon model management platform (2008), `www.eclipse.org/gmt/epsilon`

12. Kolovos, D.S., Paige, R.F., Polack, F.: The Epsilon Object Language (EOL). In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 128–142. Springer, Heidelberg (2006)
13. McDermid, J.A., Nicholson, M., Pumfrey, D.J., Fenelon, P.: Experience with the application of HAZOP to computer-based systems. In: Compass 1995: 10th Annual Conference on Computer Assurance, Gaithersburg, Maryland, pp. 37–48. National Institute of Standards and Technology (1995)
14. Simpson, H.R.: The MASCOT method. Software Engineering Journal 1(3), 103–120 (1986)
15. Wallace, M.: Modular architectural representation and analysis of fault propagation and transformation. In: FESCA 2005. ENTCS. Elsevier, Amsterdam (2005)

From Access Control Policies to an Aspect-Based Infrastructure: A Metamodel-Based Approach[⋆]

Christiano Braga[⋆⋆]

Universidad Complutense de Madrid

Abstract. Security is among the most successful applications of aspect-oriented concepts. In particular, in role-based access control, aspects capture access conditions in a quite modular way. The question we address in this paper is *how can aspects be generated from access control policies under a validated process?*

We present a metamodel-based transformation from SecureUML, a role-based access control language, to an abstract aspect language. Within this model-driven engineering context, a security policy is represented as an instance of SecureUML's metamodel and the generated aspect is represented as an instance of the abstract aspect language metamodel. Invariants specified on the *merged* metamodel of SecureUML and the abstract aspect language are checked to validate the generated aspect with respect to the given security policy.

We have prototyped our approach as a Java application on top of IT-P/OCL, a rewriting-based OCL evaluator. It outputs validated AspectJ code from a SecureUML policy.

1 Introduction

The use of aspects [14] in security is among the most successful uses of aspect-oriented concepts, both at the specification and coding levels (e.g. [19,5,11,9,18,13]).

In particular, aspects capture, in a modular way, the control conditions of role-based access control (RBAC) [10] policies. An RBAC policy describes the constraints that a given user, within a certain *role*, must fulfill in order to perform an action, that is, access, a controlled system. Essentially, access control constraints can be understood as *preconditions* to calls for controlled resources.

Preconditions can be directly represented as the so called *before advices* in aspects. Before advices are program statements that are executed before an (user-defined) identifiable execution point, or join point, in aspect-oriented terminology, is reached. An example of such a join point is a call to a particular

[⋆] Research sponsored by Ramón y Cajal program (MICINN), project DESAFIOS (TIN2006-15660-C02-01, MICINN) and project PROMESAS (S-0505/TIC/0407, CAM).

[⋆⋆] On leave from Universidade Federal Fluminense, Brasil.

method. A join point can be intercepted using a pointcut declaration in an aspect-based language, which essentially defines a pattern that matches whenever the desired join point is executed, such as a method call.

The benefits of generating aspect-code from RBAC policies is twofold: (i) we *modularly* represent access control constraints as pre-conditions, captured as before advices in an aspect, and (ii) the generated code is *automatically* called from the client code (whose access is being controlled) by the so called *weaving* process. The first benefit is actually shared with other approaches such as object-oriented design by contract. However, weaving is only supported by aspect-based languages. (See Section 2 for an example.)

The literature in the connection of aspects and RBAC is rich. In [19] the authors use aspect-oriented modeling to specify access control concerns. In [5] the focus is on web applications. The authors propose an aspect-oriented approach to declarative access control for web applications. In [11] they use aspects to implement the RBAC reference model [20]. In [9] the authors present quite clearly how aspects can be used to implement access control policies but identify deployment problems. They foresee that automating the generation process from higher-level descriptions is a must. The proposals in [18,13] research in this direction. In [18] the authors present how role-slicing models are translated into aspect code. The proposal in [13] uses aspect-oriented modeling to represent security concerns and generates aspect code.

The question we address in this paper is *how can one automatically generate aspects for access control policies under a rigorous development method, that is, within a validated generation process?*

To answer this question we propose a model driven architecture approach (MDA) [16]. Moreover, we follow the model-driven security (MDS) ideas [1]. In MDS, we quote, "designers specify system models along with their security requirements and use tools to automatically generate system architectures from the models, including complete, configured access control infrastructures." It is argued that this approach "bridges the gap between security analysis and the integration of access control mechanism into end systems. Moreover, it integrates security models with system design models and thus yields a new kind of model, *security design models*."

We have defined a transformation from role-based access control policies, modeled in the RBAC-based language SecureUML, defined in [1], to aspects, modeled in a simple abstract aspect-oriented language that we named *Aspects for Access Control* (AAC). Our aspect language essentially defines pointcuts and before advices. The aspect concepts we believe are the necessary ones to represent access control. Our approach is a metamodel-based one, that is, we consider the metamodels of the languages involved in the transformation. Therefore, a security policy is understood as an object model of the SecureUML metamodel and the generated aspect is an object model of the aspects metamodel.

The validation of the transformation process smothly integrates with the validation of SecureUML policies proposed in [6]. There, SecureUML policies are validated by evaluating the OCL *invariants* of the SecureUML metamodel over policies captured as object models. We extend their approach, in the context of MDA, by proposing that model transformations from SecureUML policies to code could be validated on the *merged* metamodel of SecureUML and target language. Therefore, we consider the merged metamodel of SecureUML and AAC, our aspect laguage, to guarantee conformance of a generated aspect with respect to the given security policy. We name *transformation invariants* the invariants associated with the merged metamodel.

Our MDA approach applies the metamodel transformation and model merging approaches from [16]. Moreover, it neither requires a new specification language, such as QVT [17], for its specification, nor commits to a particular implementation. A fact that we believe to be positive in the sense of using the same languages from the modeling phase (UML and OCL) and not imposing a particular implementation framework. A point of view that we believe is shared with [2]. At the implementation level, any programming language can be used as long as the implementation preserves the transformation invariants.

We have prototyped our approach on top of the rewriting-based OCL evaluator ITP/OCL [8,3]. The prototype is implemented in Java and essentially loads ITP/OCL with all the above mentioned metamodels and invariants. When given an object model of a security policy as input the transformer generates an object model of an aspect. The invariants of the merged metamodel are then evaluated on the union of the object models of security policy and aspect, following the Design by Contract [15] idea of run-time monitoring of assertions. If they hold, the transformer outputs an aspect in AspectJ syntax.

To summarize our contribution in one paragraph: we propose an *automatic* and *validated* code generation process from role-based access control policies into aspect code. Each policy gives rise to an aspect. The validation of the aspect generation is model-based. The generated aspect is validated by evaluating the OCL invariants, defined for the model of SecureUML and aspects, on model instances that represent the security policy and aspect. We have prototyped our approach on top of an OCL evaluator.

This paper is organized as follows. In Section 2 we illustrate how a SecureUML policy may be represented by an aspect. Section 3 presents our metamodel-based transformation approach to aspect generation from SecureUML access control policies. Section 4 discusses a prototype implementation to our approach. Finally, Section 5 concludes this paper with final remarks.

2 From SecureUML Policies to Aspect Code

An access control policy in SecureUML specifies which (user) *roles* are given *permissions* to perform *actions* over *resources* under certain *authorization constraints*. SecureUML has a loose semantics, in an algebraic sense [12], with respect to the resources which access may be controlled. Our resource language of choice, called ComponentUML, defines resources to be entities which may have

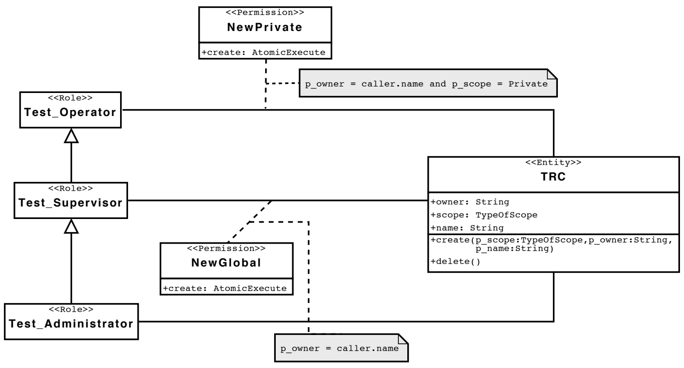


Fig. 1. An Example Access Control Policy

attributes, methods, and association ends. Attributes, methods, and association ends are also resources which access may be controlled.

As an example, consider the SecureUML policy specified by the model in Figure 1. The model specifies the access control over the entity *TRC* (for test report configuration). Users under three different roles may access a *TRC*: *Test_Operator*, *Test_Supervisor* or *Test_Administrator*. These roles are related under role inheritance. The most "powerful" role is a *Test_Administrator*, who inherits the access controls permissions from the *Test_Supervisor* role. The latter inherits from the *Test_Operator* role. Therefore, the *Test_Operator* role is the less powerful one.

For a user to execute the (atomic) *create* action over *TRC*, under a *Test_Operator* role, then the authorization constraint attached to the permission *NewPrivate* must hold. This permission means that the parameter *p_owner* of the *create* action must be equal to the name of the user (denoted by SecureUML's special variable *caller*) and also that the parameter *p_scope* of the *create* action must be equal to the constant *Private*. To perform the same action under the role of *Test_Supervisor* the permissions *NewGlobal* or *NewPrivate* (since *Test_Supervisor* inherits permissions from *Test_Operator*) must hold.

The model in Figure 1 is a quite simplified version of the models produced in an industrial project with a major Spanish technology company, which results are reported in [7].

From this model, essentially, we produce an abstract class, that represents an interface to a concrete implementation of the *TRC* entity, and an aspect, that implements the access control to *TRC*'s methods. For each *TRC* method, only *create* in this example, we declare, in the *TRCAccessControl* aspect, a pointcut, identified by *createPC*, to capture a call to *create*, and a before advice. The advice is triggered whenever *create* is called and checks if the authorization constraints for *NewGlobal* or *NewPrivate* hold. It raises an exception otherwise. The abstract class and aspect for the *TRC* access control policy in Figure 1 is presented below.

```
abstract class TRC {
   int scope ;    String owner ;    String name ;
   abstract TRC create( int p_scope, String p_owner, String p_name )
                                                    throws Exception ;
}

aspect TRCAccessControl {
   static Role caller ;
   TRCAccessControl() { caller  = Env.getUserRole() ; }
   pointcut createPC(int s, String o, String n) :
     call(TRC TRC.create(int, String, String)) && args(s,o,n) ;
   before(int s, String o, String n) throws Exception : createPC(s,o,n)
   {
       if (caller  instanceof TestSupervisor)
          if (o == caller.name) return ;
          else throw new
              Exception("Current user may not create a TRC.") ;
       if (caller  instanceof TestOperator)
          if ((s == TypeOfScope.PRIVATE) && (o == caller.name)) return ;
          else throw new
              Exception("Current user may not create a TRC.") ;
       throw new Exception("Current user has role "+caller+
                                          " and may not create a TRC.") ;
   }
}
```

In our example, whenever the method `create`, from the class *TRC*, is called (represented by pointcut `createPC`), the associated before advice will be executed before the body of the method `create`. Therefore, the aspect modularly captures the access control requirement and the aspect weaving process transparently integrates it with the client code, which, in our example, calls method `create` from class `TRC`.

We have implemented an automatic and validated process for generating aspect code from SecureUML policies. Our translation follows the MDS ideas mentioned in the introductory section. The transformation process is the subject of Section 3.

3 Transforming SecureUML Policies into Aspects

Our approach consists on a validated and automatic transformation from SecureUML policies to aspect code. The target language of our transformation is an abstract aspect-oriented language, that we call Aspects for Access Control (AAC) which has abstract classes, protected attributes, methods, aspects, pointcuts and before advices as language constructs; the elements that appear to be necessary from the aspect-oriented paradigm to code access control.

The proposed transformation is thus a transformation between domain-specific languages: the source captures access control policies and the target an aspect language for access control. The transformation function essentially relates each component that requires access control security (called an entity in SecureUML terminology) into an abstract class and an aspect. The abstract class represents an interface that a concrete implementation component of the controlled component must implement. The aspect implements the access control constraints that must hold when a component's method (overloaded from the associated abstract class method) is called.

We use metamodels to specify and validate the transformation process, following the MDA approach and aiming at a smooth integration of the transformation process into the modeling phases. The syntax of each language (i.e. SecureUML and AAC) is specified as a metamodel in UML, together with its OCL invariants that capture the structural constraints of the given metamodel. We extend the model-based validation process for SecureUML proposed in [6] by defining the metamodel of SecureUML merged with AAC that disjointedly unites the two languages, adds new relationships among their classes and new invariants over the merged language. It specifies when an AAC model is a properly generated one from a given security policy.

In our proposal, validation means to check the invariants for a given access control policy, for the generated abstract class and aspect and all of them together, that is, for the generated abstract class and aspect with respect to the given access control policy. The validation process should occur in two different moments in time:

1. Before the transformation is applied: the invariants of the SecureUML metamodel are applied to the given access control policy. This step guarantees that the given access control policy is well-formed.
2. After the transformation is applied:
   (a) The invariants of the AAC metamodel are applied to the generated abstract class and aspect. This step guarantees that they form a valid AAC model *per se*.
   (b) The invariants of the *merged* metamodel of SecureUML and AAC are applied to both the security policy and the generated abstract class and aspect.

The validation process is automatic and is implemented on top of ITP/OCL tool, an OCL evaluator. This is the subject of Section 4.

This section continues as follows: in Section 3.1 we discuss the metamodels for each language and in Section 3.2 we outline the transformation function from SecureUML to AAC.

3.1 A Metamodel-Based Approach

A metamodel defines the elements of a language, relationships between elements of a language and assertions that constraint the relationships between the elements of a language.

We chose UML's class diagrams to specify a language. Therefore, a metamodel essentially consists of classes, attributes, methods, associations between classes

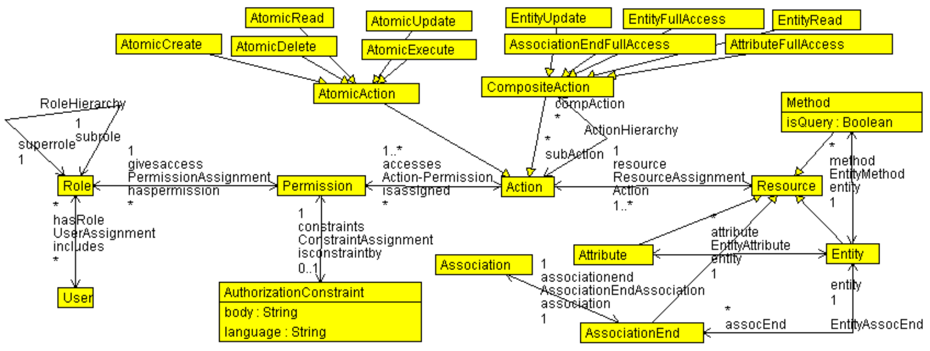


Fig. 2. The SecureUML+ComponentUML metamodel

(with roles and multiplicities) and generalizations between classes. OCL is our specification language of choice to specify invariants on metamodels.

SecureUML Metamodel. SecureUML[1] provides a language for modeling *Roles*, *Permissions*, *Actions*, *Resources*, and *Authorization Constraints*, along with their *Assignments*, i.e., which permissions are assigned to which roles, which actions are assigned to which permissions, which resources are assigned to which actions, and which constraints are assigned to which permissions. In addition, actions can be either *Atomic* or *Composite*. The atomic actions are intended to map directly onto actual operations of the modeled system. The composite actions are used to hierarchically group more lower-level ones and are used to specify permissions for sets of actions.

SecureUML leaves open what the protected resources are and which actions they offer to clients. These are specified in a so-called *dialect* and depend on the primitives for constructing models in the system design modeling language of choice. ComponentUML is our dialect of choice. It is a simple language for modeling component-based systems. Essentially, it provides a subset of UML class models: *Entities* can be related by *Associations* and may have *Attributes* and *Methods*. Therefore, by using SecureUML+ComponentUML, it is possible to model the permissions that an user playing a given role has over an entity, an attribute, a method or an associations, i.e., the actions such an user can execute while trying to access the resource. The metamodel of SecureUML+ComponentUML is given in Figure 2. Note that the security policy drawn in Figure 1 shows an instance of SecureUML+ComponentUML metamodel.

AAC Metamodel. The elements of the AAC metamodel are: *ResClass* (where the prefix *Res* stands for resource), *ResAttribute*, *ResMethod*, *ResGetMethod*, *ResSetMethod*, *Aspect*, *Pointcut*, *BeforeAdvice*, *RoleClass* and *Env*. The metaclasses *ResClass*, *ResAttribute* and *ResMethod* represent elements of the generated abstract class after the application of the transformation function to a

[1] The material in this subsection is adapted from [7].

SecureUML policy. A *ResMethod* has two subclasses: *ResGetMethod* and *ResSetMethod*. The metaclasses *Aspect*, *Pointcut* and *BeforeAdvice* represent elements of the generated aspect. The metaclass *RoleClass* represents *Roles* as classes. The metaclass *Env* represents the environment that must provide user run-time information such as the user's current role and the user's name.

An instance of *ResClass* may have many *ResAttributes* and *ResMethods*. An instance of an *Aspect* may have many *Pointcuts* and each *Pointcut* has one and only one *BeforeAdvice*. Moreover, an instance of an *Aspect* must be related with one and only one *ResClass* and each *Pointcut* must be associated with a single *ResMethod*. The metaclass *RoleClass* is related to itself.

An example of OCL invariant for this metamodel is that each *ResMethod* in a *ResClass* must have one *Pointcut* in the *Aspect* associated with the *ResClass*. This invariant could be called consistency between *ResMethod* and *Pointcut*. The OCL invariant below declares that the navigation from a *ResMethod* through its link to its *Pointcut* and then its *Aspect* should point to the same object as navigating through the *ResMethod*'s *ResClass* and then its *Aspect*.

```
context ResMethod inv:
self.allInstances()−>forall(rm |
rm.Pointcut-ResMethod.Aspect-Pointcut = rm.Class-ClassMethod.Aspect-Class)
```

The Merged Metamodel. The classes in the merged metamodel of SecureUML and AAC are given by the disjoint union of the classes in the metamodel of SecureUML and the classes in the metamodel of AAC. The relations in the merged metamodel of SecureUML and AAC are given by the disjoint union of the relations on each metamodel including new relations that associate the classes on each metamodel. Moreover, it specifies which properties must hold so that an instance of the merged metamodel of SecureUML and AAC is well-formed. That is, if an AAC model is a valid abstract class and aspect for the given SecureUML policy, with respect to the invariants defined in the merged metamodel.

In the merged metamodel, an *Aspect* is associated with one and only one *Entity* and such metaclass is related to a single *ResClass* to represent the class of an entity. A *ResMethod* is associated with one and only one *Method*, and a *ResAttribute* is associated with a single *Attribute*. The subclasses of *ResMethod*, *ResGetMethod* and *ResSetMethod*, are related to *Attribute* in order to indicate the getters and setters of the attributes, if any. In addition, an *AuthorizationConstraint* is associated with one and only one *BeforeAdvice* that will implement the constraint but such an advice may include the implementation of several constraints.

An example of an OCL invariant over the merged metamodel is shown below.[2] The invariant specifies that for each *Method*, there exists a *ResMethod* in the *ResClass* associated with the *Method*'s *Entity*, such that, for each *ResMethod*, the *BeforeAdvice* in the *ResMethod*'s *Pointcut* is associated with the *ResMethod*'s *Authorization Constraints*.

[2] The complete set of invariants for the merged metamodel can be found in `http://maude.sip.ucm.es/~cbraga/transformationInvariants.pdf`.

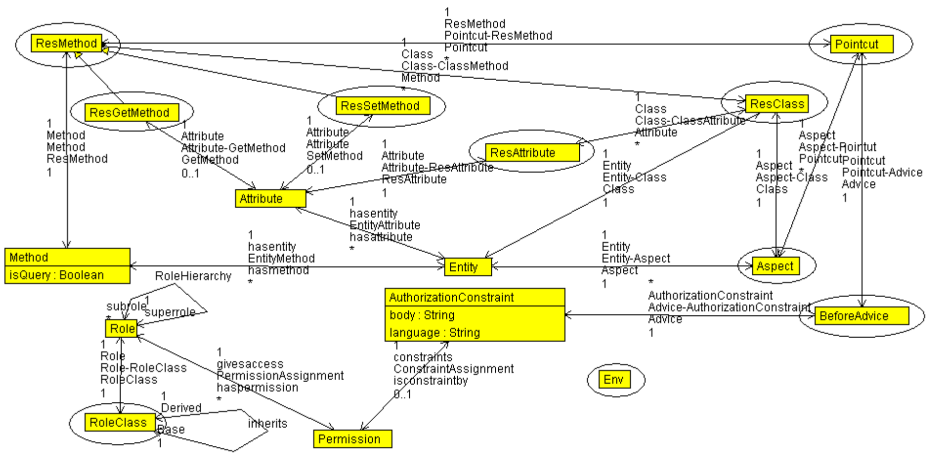


Fig. 3. A subset of the merged metamodel

The invariant uses two auxiliary operations:
(i) *ResMethod::validateResMethodConstraints(Method m):Boolean* and
(ii) *Method::getMethodResMethods:Set(ResMethod)*. The second one, which declaration is not shown, returns the *ResMethods* associated with a given *Method*. The first operation, shown below, checks if the given *Method* is associated with the current *ResMethod* (denoted by the special OCL variable *self*) and if the *AuthorizationConstraints* in the *BeforeAdvice* associated with the *Method*'s *Pointcut* are the same as the given *Method*'s *AuthorizationConstraints*, returned by the function *Method::allMethodConstraints()*.

```
context Method inv:
self.allInstances()−>forAll(m | (m.getMethodResMethods())−>
                    exists(rm | rm.validateResMethodConstraints(m)))

context ResMethod::validateResMethodConstraints(m:Method):Boolean body:
(self.Method = m) and
(self.Pointcut-ResMethod.Pointcut-Advice.Advice-AuthorizationConstraints =
 m.allMethodConstraints())
```

3.2 The Transformation Function

For a given SecureUML policy the transformation function produces for each *Entity* an abstract class and an aspect. The abstract class (*ResClass*) represents the interface that has to be fulfilled by a concrete implementation for the given *Entity*. This abstract class is comprised by attributes (*ResAttribute*) and methods (*ResMethod*) that represent their *Entity*'s counterparts in the SecureUML policy and moreover: (i) with all attributes declared with protected visibility (that is, only directly accessible by instances of the given class or its heirs) and

(ii) with the so called "getters" and "setters" methods for each attribute, that is, methods to, respectively, read and update the state of each attribute.

The *Aspect* generated by the transformation function controls the calls to the methods of the generated *ResClass*. For each *ResMethod* there exists a *Pointcut* and a *BeforeAdvice*. The *Pointcut* is declared as a call to the given *ResMethod*. The *BeforeAdvice* implements the permissions of the resource associated with the *ResMethod*, as follows:

- If the given *ResMethod* is a "getter" method to a *ResAttribute* then the body of the *BeforeAdvice* implements the *AuthorizationConstraints* of the read permissions of the attribute associated with the *ResAttribute* guarded by the given *ResMethod*. (Read permissions are those related with *Atomic Read*, *Attribute Full Access*, *Entity Read* and *Entity Full Access* actions in a SecureUML policy.)
- If the given *ResMethod* is a "setter" method to a *ResAttribute* then the body of the *BeforeAdvice* implements the *AuthorizationConstraints* of the write permissions of the *Attribute* associated with the *ResAttribute* controlled by the given *ResMethod*. (Write permissions are those related with *AtomicUpdate*, *AtomicDelete*, *Attribute Full Access*, *Entity Update* and *Entity Full Access* actions in a SecureUML policy.)
- If the given *ResMethod* is associated with a *Method* then the body of *BeforeAdvice* implements the *AuthorizationConstraints* of the permissions of the *Method* associated with the given *ResMethod*.

The *AuthorizationConstraints* are essentially predicates over the state of their associated *Entity*. We assume, for the sake of simplicity of this explanation, that each of them is already coded in the concrete syntax of our target language. The implementation of an *AuthorizationConstraint* is a condition which first tests for the user's *Role*, with respect to the *AuthorizationConstraint*'s *Role*, and then checks for the *AuthorizationConstraint*'s predicate.

The body of a *BeforeAdvice* is essentially a sequence of conditions. It may return successfully (then allowing a *ResMethod* to be called) if the user has an appropriate *Role* and fulfills the *AuthorizationConstraints* of at least one of the *Permissions* associated with the given *ResMethod*. Otherwise it returns an error (for instance, by raising an exception) if no *Permission* is fulfilled or if the user does not have an appropriate *Role* that copes with any of the *Permissions*. Therefore, the conditions in the body of a *BeforeAdvice* are *ordered* by the *Role* associated with the *AuthorizationConstraint* that the condition implements, starting from the most powerful one (e.g. an administrator) to the least powerful one (e.g. an operator).

Each *Role* is transformed into a *RoleClass*. The *Role* hierarchy relationship is captured as a *RoleClass* inheritance relationship.

4 Monitoring Transformation Invariants

We have implemented the transformation function described in Section 3.2 as a prototype Java application on top of the OCL evaluator ITP/OCL. Our implementation is a three-tiered application. The higher layer implements the transformation function, the middle layer is a Secure UML policy manager and the bottom layer is an OCL evaluator.

The OCL evaluator is a "wrapper" Java class that provides access to IT-P/OCL. Essentially, it defines methods for:

– Creation of class and instance diagrams.
– Creation and deletion of classes, relationships between classes, objects and links between objects.
– Evaluation of OCL queries on instance diagrams.
– Evaluation of OCL invariants on instance diagrams.

The SecureUML policy manager is implemented as a Java class that instantiates the OCL evaluator with the SecureUML metamodel as class diagram. The SecureUML policy manager defines an API to create SecureUML policies and operations to query a SecureUML policy. A SecureUML policy is represented internally as an instance diagram of the SecureUML metamodel. Operations on a SecureUML policy are translated into OCL expressions and executed as queries in the OCL evaluator instance held by the SecureUML policy manager. Moreover, it implements all the invariants and operations over the SecureUML metamodel defined in [6].

The SecureUML to aspect transformer is implemented as a Java class that extends the SecureUML policy manager. Given a SecureUML policy, the transformer instantiates a set of Java classes that faithfully represent the AAC metamodel described in Section 3.1. The instantiation process follows the transformation defined in Section 3.2. The set of Java objects produced in memory by the transformer are then traversed in order to generate an object model of AAC in the underlying instance of the OCL evaluator. Finally, all the invariants are checked and the abstract class and aspect are written into the output using AspectJ syntax together with Java classes for each role with the appropriate inheritance relationship.

The metaclass *Env* is translated to a Java class named *Env* with a single method with signature `Role getUsrRole()`. This method returns the role of the current user. The prototype generates a simple implementation for `getUserRole` just for the purpose of our experiment. This implementation is shared with the application that the generated aspect is connected with. Of course, a more robust component could have been targeted that takes advantage of user information from the underlying operational system, for example.

Note that the function that produces the concrete syntax in AspectJ could be *overloaded* to produce different concrete syntax for Design by Contract languages such as Eiffel, JML or Spec#. For these languages the generated abstract class would be annotated with preconditions representing the access control assertions. Of course, in this case, the weaving process is not automatic and explicit calls to the methods implementing the pre-conditions would have to be written.

5 Final Remarks

The use of aspects in security is among the most successful uses of the AOP paradigm. In particular, we refer to [18,13,19] as representatives of the use of AOP in access control.

In [18] the authors formalise *role-slices* to specify access control policies and a compilation process from access control policies into aspect code. The compilation process is described as a functional program. The process appears to be quite precise however there is no indication of a correctness proof or any validation of the proposed process. In [19] the authors propose the use of aspects at the modeling level (similarly to [13]) but analysis is left to future work.

The approach followed in [13], we quote, "translates security aspects specified as UMLsec stereotypes as concrete security mechanisms on the modelling level". They analyze, with a theorem prover, first-order logic (FOL) formulae generated out of control flow graphs obtained from the produced source code and associated security requirements. Their approach appears to be similar to [1,4] where the target of the transformation process is a FOL theory that can be reasoned about. Our approach is part of the so called *lightweight* formal methods. We aim at *validation* instead of the *verification* approach in [13,1,4]. Our approach allows the validation of each and every instance of the application of the transformation function over an access control policy and generated (abstract class and) aspect. We do not aim at allowing for the proof of general (inductive) properties that would require theorem proving but rather an approach following the pragmatic ideas of Design by Contract (DbC). In [1] the authors propose two transformations from SecureUML policies targeting two different object-oriented frameworks. We complement that effort with another transformation to aspects that further extends the code generation ideas in [6,7]. Moreover, we extend [6] by applying validation to transformation invariants.

What appears to be novel in our MDS approach from RBAC policies to code is to generate *validated* AspectJ code. The use of aspects *modularly* code permission's constraints. We validate our transformation using a *merged* metamodel of the abstract syntax of the languages involved, focusing on the transformation invariants that specify structural constraints that the implementation of the transformation has to cope with. We do not commit ourselves to any particular transformation specification language or implementation language. The specification of the transformation occurs under the same model-based approach used during design. The transformation invariants are then checked by a tool during runtime, following DbC's run-time assertion monitoring idea. Also, the generation of different concrete syntax, besides AspectJ, can be targeted such as a DbC-based language.

We believe that our approach thus contributes to the efforts both of code generation for model driven security [1] and perhaps to model driven architecture itself, in the context of [2]. We foresee the continuation of this work with more experimentation, in particular on the exploitation of the notion of transformation contracts over merged metamodels, and by enhancing our tool support for OCL evaluation and SecureUML policy manager, both in terms of efficiency.

Acknowledgements. I would like to thank Viviane Silva and Marina Egea for their comments on a draft of this paper. Jorge Ogalla Ramírez and Pedro Díaz Yeregui work on the implementation on the prototype is also acknowledged.

References

1. Basin, D.A., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Transactions on Software Engineering and Methodology 15(1), 39–91 (2006)
2. Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A.: Model transformations? Transformation models! In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 440–453. Springer, Heidelberg (2006)
3. Braga, C., Clavel, M., Durán, F., Eker, S., Farzan, A., Hendrix, J., Lincoln, P., Martí-Oliet, N., Meseguer, J., Olveczky, P., Palomino, M., Sasse, R., Stehr, M.-O., Talcott, C., Verdejo, A.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350, pp. 667–693. Springer, Heidelberg (2007), `http://dx.doi.org/10.1007/978-3-540-71999-1_21`
4. Brucker, A.D., Doser, J., Wolff, B.: A model transformation semantics and analysis methodology for secureUML. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 306–320. Springer, Heidelberg (2006)
5. Chen, K., Lin, C.-W.: An aspect-oriented approach to declarative access control for web applications. In: Zhou, X., Li, J., Shen, H.T., Kitsuregawa, M., Zhang, Y. (eds.) APWeb 2006. LNCS, vol. 3841, pp. 176–188. Springer, Heidelberg (2006)
6. Clavel, M., Basin, D., Doser, J., Egea, M.: Automated analysis of security-design models. Information and Software Technology (2008), `http://maude.sip.ucm.es/~clavel/pubs/BCDE07-journal.pdf`
7. Clavel, M., da Silva, V., Braga, C., Egea, M.: Model-driven security in practice: An industrial experience. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 326–337. Springer, Heidelberg (2008), `http://maude.sip.ucm.es/~clavel/pubs/CSBE08.pdf`
8. Clavel, M., Egea, M.: ITP/OCL: A rewriting-based validation tool for UML+OCL static class diagrams. In: Johnson, M., Vene, V. (eds.) AMAST 2006. LNCS, vol. 4019, pp. 368–373. Springer, Heidelberg (2006)
9. de Win, B., Vanhaute, B., Decker, B.D.: Security through aspect-oriented programming. In: Proceedings of the IFIP TC11 WG 11.4 First Annual Conference on Netwrok Security: Advances in Network and Distributed Systems Security, vol. 206, pp. 125–138 (2001)
10. Ferraiolo, D.F., Kuhn, D.R., Chandramouli, R.: Role-Based Access Control, 2nd edn. Artech House Publishers (2007)
11. Gao, S., Deng, Y., Yu, H., He, X., Beznosov, K., Cooper, K.: Applying aspect-orientation in designing security systems: A case study. In: Proceedings of 16th International Conference on Software Engineering and Knowledge Engineering, Banff, Alberta, Canada, June 20-24, pp. 360–365 (2004)
12. Goguen, J.A., Meseguer, J.: Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theoretical Computer Science 105(2), 217–273 (1992)
13. Jürjens, J., Houmb, S.H.: Dynamic secure aspect modeling with UML: From models to code. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 142–155. Springer, Heidelberg (2005)

14. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of aspectJ. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, pp. 327–355. Springer, Heidelberg (2001)
15. Meyer, B.: Object-Oriented software construction, 2nd edn. Prentice-Hall, Englewood Cliffs (1997)
16. Miller, J., Mukerji, J. (eds.): MDA Guide (Version 1.0.1). Number omg/2003-06-01. OMG (2006)
17. Object Management Group. MOF QVT Final Adopted Specification, OMG Adopted Specification ptc/05-11-01 (2005)
18. Pavlich-Mariscal, J.A., Michel, L., Demurjian, S.A.: A formal enforcement framework for role-based access control using aspect-oriented programming. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 537–552. Springer, Heidelberg (2005)
19. Ray, I., France, R., Li, N., Georg, G.: An aspect-based approach to modeling access control concerns. Information and Software Technology 46(9), 575–587 (2004)
20. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer 29(2), 38–47 (1996)

Eighth International Workshop on OCL Concepts and Tools

Jordi Cabot[1], Martin Gogolla[2], and Pieter Van Gorp[3]

[1] Open University of Catalonia Spain
jcabot@uoc.edu
[2] University of Bremen Germany
gogolla@informatik.uni-bremen.de
[3] University of Antwerp Belgium
pieter@pietervangorp.com

Abstract. This paper reports on the 8th OCL workshop held at the MODELS conference in 2008. The workshop focussed on how to evaluate, compare and select the right OCL tools for a given purpose and how to deal with the expressiveness and complexity of OCL. The workshop included sessions with paper presentations as well as a special tool demo session.

1 Introduction

In recent years, MDA and associated MDE methodologies, approaches and languages (like QVT) emphasized the role that the Object Constraint Language (OCL) has to play in MDE development. Moreover, the modeling community is continuously pushing forward the OCL, far beyond its initial requirements as a precise modeling language complementing UML descriptions. Now, OCL is used in quite different applications domains (e.g., domain-specific languages and web semantics) and for various purposes (e.g., model verification and validation, code-generation, test-driven development, and transformations). To be successful, all these new OCL applications, extensions and usages require new OCL tools that support them.

This workshop aimed to look specifically at how to develop, apply, evaluate and compare all kinds of OCL-related tools. The workshop brought together OCL practitioners and OCL tool builders (from both academy and industry) in order to evaluate today's state-of-the-practice. In particular, the workshop focused the discussion on how to evaluate, compare and select the right OCL tools for a given purpose, how to deal with the expressiveness and complexity of the language and how to tackle its ambiguous or underdefined issues from a practical point of view. The workshop discussed new OCL tools and patterns, libraries, and algorithms that may facilitate development and reuse. In addition, all other aspects that may improve the adoption and support of OCL or its usability were considered. As a result, the workshop contributed to consolidate and expand the role of OCL in the modeling community by discussing approaches for research and development that could potentially drive the building of new OCL tools.

All submitted papers were reviewed by three industrial or academic members from the Program Committee:

– David H. Akehurst, Thales, UK
– Thomas Baar, Tech@Spree, Germany
– Jean Bézivin, University of Nantes, France
– Behzad Bordbar, University of Manchester, UK
– Achim Brucker, SAP, Germany
– Dan Chiorean, University of Cluj, Romania
– Tony Clark, Ceteva, UK
– Birgit Demuth, Technical University of Dresden, Germany
– Remco Dijkman, Eindhoven University of Technology, The Netherlands
– Robert France, University of Fort Collins, USA
– Heinrich Hussmann, University of Munich, Germany
– Richard Mitchell, Inferdata, UK
– Mark Richters, Astrium Space Transportation, Germany
– Shane Sendall, IBM, Switzerland
– Burkhart Wolff, University of Paris-Sud (Orsay), France
– Steffen Zschaler, Lancaster University, UK

2 Workshop Papers

Here we give a short overview of the papers presented in the workshop taken from the abstracts provided by the authors. The papers were organized in three different sessions: *Executing OCL* (papers 1 to 3), *Translating OCL* (4 to 6) and *Validation and Verification* (7 to 9). Polished versions of the workshop papers will appear in an EASST special issue.

2.1 Building an Efficient Component for OCL Evaluation by Manuel Clavel, Marina Egea and Miguel García de Dios

In this paper we report on our experience implementing the Eye OCL Software (EOS) evaluator, a Java component for efficient OCL evaluation. We first motivate the need for an efficient implementation of OCL in order to cope with novel usages of the language. We then discuss the performance of the EOS component when evaluating expressions on medium-large scenarios. Finally, we explore various approaches for evaluating OCL expressions on really large scenarios.

2.2 Static Source Code Analysis Using OCL by Mirko Seifert and Roland Samlaus

The majority of artifacts created during software development are representations of programs in textual syntax. Although graphical descriptions are becoming more widespread, source code is still indispensable. To obtain programs that behave correctly and adhere to given coding conventions, source code must be analyzed — preferably using automated tools. Building source code analyzers has a long tradition and various mature tools exist to check code written in conventional languages, such as Java or C. As new languages emerge (e.g., Domain Specific Languages) these tools can not be applied and building a tool for each language does not seem feasible either. This paper investigates how meta models for textual languages and the Object Constraint Language can enable generic static source code analysis for arbitrary languages. The presented approach is evaluated using three languages (Java, SQL and a DSL for state machines).

2.3 Optimization Patterns for OCL-Based Model Transformations by Jesús Sánchez Cuadrado, Frédéric Jouault, Jesús García-Molina and Jean Bézivin

Writing queries and navigation expressions in OCL is an important part of the task of developing a model transformation definition. When such queries are complex and the size of the models is significant, performance issues cannot be neglected. In this paper we present five patterns intended to optimize the performance of model transformations when OCL queries are involved. For each pattern we will give an example as well as several implementation alternatives. Experimental data gathered by running benchmarks is also shown to compare the alternatives.

2.4 An Incremental OCL Compiler for Modeling Environments by Tamás Vajk and Tihamer Levendovszky

In software engineering, reliability and development time are two of the most important aspects, therefore, modeling environments, which aide both, are widely used during software development. UML and OCL became industry standards, and are supported by many CASE tools. OCL code checking, which has to be performed by these tools, has a specialty, as not all of the information necessary for compilation is available from the code, the related model contains the types, navigations and attributes. The build time of OCL code fragments is increased if the development tool supports distributed modeling, because in this case, model element checking has to be performed in a model repository that cannot be held in memory. In this paper, we introduce a method that enables incremental OCL code building and therefore reduces the development time. Incremental builds require higher complexity than simple builds, thus balancing between the two methods is also considered.

2.5 Implementing Advanced RBAC Administration Functionality with Use by Tanveer Mustafa, Karsten Sohr, Duc-Hanh Dang, Michael Drouineaud and Stefan Kowski

Role-based access control (RBAC) is a powerful means for laying out and developing higher-level organizational policies such as separation of duty, and for simplifying the security management process. One of the important aspects of RBAC is authorization constraints that express such organizational policies. While RBAC has generated a great interest in the security community, organizations still seek a flexible and effective approach to impose role-based authorization constraints in their security-critical applications. In particular, today often only basic RBAC concepts have found their way into commercial RBAC products; specifically, authorization constraints are not widely supported. In this paper, we present an RBAC administration tool that can enforce certain kinds of role-based authorization constraints such as separation of duty constraints. The authorization constraint functionality is based upon the OCL validation tool USE. We also describe our practical experience that we gained on integrating OCL functionality into a prototype of an RBAC administration tool that shall be extended to a product in the future.

2.6 Shortcomings of the Embedding of OCL into QVT ImperativeOCL by Fabian Büttner and Mirco Kuhlmann

MOF QVT introduces ImperativeOCL as an imperative language for operational descriptions of model transformations (QVT operational mappings). ImperativeOCL extends conventional OCL by expressions with side-effects. A couple of semantical problems arise from the way OCL is embedded into ImperativeOCL imperative expressions are modelled as a subtype of OCL expressions. This paper points out these semantical problems and proposes a change to the operational mappings language of QVT that resolves these problems, following an approach that reuses OCL by composition rather than by inheritance in the abstract syntax of ImperativeOCL. The proposed change reduces the complexity of the imperative language, removes undefinedness, and leaves OCL conformant to its original definition.

2.7 Observations for Assertion-Based Scenarios in the context of Model Validation by Emine Aydal, Richard Paige and Jim Woodcock

Certain approaches to Model-Based Testing focus on test case generation from assertions and invariants, e.g., written in the Object Constraint Language. In such a setting, assertions and invariants must be validated. Validation can be carried out via executing scenarios wherein system operations are applied to detect unsatisfied invariants or failed assertions. This paper aims to improve our understanding of how to write useful validation scenarios for assertions in OCL. To do so, we report on our experiences during the creation and execution of 237 scenarios for validating assertions for the Mondex Smart Card application. We also describe key factors that must be considered in transforming scenarios into test cases.

2.8 Executing Underspecified OCL Operation Contracts with a SAT Solver by Matthias P. Krieger and Alexander Knapp

Executing formal operation contracts is an important technique for requirements validation and rapid prototyping. Current approaches require additional guidance from the user or exhibit poor performance for underspecified contracts that describe the operation results non-constructively. We present an efficient and fully automatic approach to executing OCL operation contracts which uses a satisfiability (SAT) solver. The operation contract is translated to an arithmetic formula with bounded quantifiers and later to a satisfiability problem. Based on the system state in which the operation is called and the arguments to the operation, an off-the-shelf SAT solver computes a new state that satisfies the postconditions of the operation. An effort is made to keep the changes to the system state as small as possible. We present a tool for generating Java method bodies for operations specified with OCL. The efficiency of our method is confirmed by a comparison with existing approaches.

2.9 How My Favorite Tool Supporting OCL Must Look Like by Dan Chiorean, Vladiela Petrascu and Dragos Petrascu

This paper presents its authors' viewpoint on the assessment of tools that support the use of OCL. At this time, deciding on which such tool to use is not an easy task. This is influenced by a number of objective factors, including: the user's needs, knowledge of existing tools, knowledge of the language and of the various possibilities of using it. Undoubtedly, the choice of a particular tool does also include subjective aspects. The paper is limited to the presentation of objective criteria. In this context are examined: the features that distinguish OCL within the modeling languages' family, some aspects that are either incompletely or ambiguously described in the OCL specification, the main functionalities that an OCL supporting tool should implement, the universe of tools supporting OCL. In the end, the tools listed in Section 6 are characterized with respect to the functionalities of an ideal tool and the obtained conclusions are presented.

3 Tool Showcase

The following tools were demonstrated during a two hour session at the end of the workshop: RoclET[1], The Epsilon Validation Language[2], The HOL-OCL system [3], UMLtoCSP[4], The Eye Secure Software (ESS[5]), UML Specification En-

[1] http://www.roclet.org/ by Thomas Baar, Cédric Jeanneret and Slavisa Markovic.
[2] http://www.eclipse.org/gmt/epsilon/ by Dimitrios Kolovos and Richard Paige.
[3] http://www.brucker.ch/projects/hol-ocl/ by Achim D. Brucker and Burkhart Wolff.
[4] http://gres.uoc.edu/UMLtoCSP/ by Robert Clarisó, Jordi Cabot and Daniel Riera.
[5] http://maude.sip.ucm.es/eos/ by Miguel A. García de Dios, Marina Egea and Manuel Clavel .

vironment (USE[6]), RestrictED[7], Advanced OCL Editor based on Eclipse OCL[8], Visual Modeling and Transformation System (VMTS[9]), OCLE[10] and Kermeta[11]. Most of these demonstrations have been recorded by specialized screen capture software and can be downloaded from the workshop website[12].

4 Lessons Learned

The workshop triggered a series of interesting discussion threads, most of which may guide future work on OCL related tools. First of all, several questions and remarks related to whether OCL tools should be extended with OCL specific algorithms (analyses, optimizations, ...) or whether one should translate OCL into other languages for those purposes. For example, in the context of the performance evaluation of Clavel et al., several attendees proposed to leverage existing work on query optimization from the database domain. Others were convinced that several applications required optimizations that could only be realized within the evaluator of an OCL tool itself. In the end, it remained an open issue whether it would be most promising to focus on *bridging technological spaces* or on *improving the OCL/modeling space*, perhaps by reusing (probably specializing) expertise from other spaces.

In another discussion thread, one questioned whether or not the OCL community was mature enough to focus on performance. In this case, Jouault demonstrated that the semantics of the OCL collection operations was clear enough to reason about correctness while internally changing the mutability of collections for performance reasons. The participants agreed that one should not extend the OCL with performance specific language constructs: the language should stimulate conceptual modeling, and a modeler should not have a particular tool in mind. Obviously, it may be much easier for tool builders to expect from users that a particular style of specification is used. However, tool builders should go further and rewrite specifications that focus on non-technical issues into more technical OCL specifications that are ready for efficient execution.

Finally, the participants expressed concerns about communication channels to the authors of the OCL standard. More specifically, the community wants a light-weight communication channel to discuss problems about the standard. Summarizing such problems solely in scientific articles may be insufficient and sometimes even inappropriate.

[6] http://www.db.informatik.uni-bremen.de/projects/USE/ by Fabian Büttner.
[7] http://www.inf.tu-dresden.de/~ms72/RestrictED/ by Mirko Seifert and Roland Samlaus.
[8] http://squam.info/ocleditor/ by Joanna Chimiak-Opoka, Ekrem Arslan, Hannes Moesl and Franz-Josef Peer (and presented by Michael Felderer.)
[9] http://vmts.aut.bme.hu/ by Tamas Vajk, Gergely Mezei and Tihamer Levendovszky.
[10] http://lci.cs.ubbcluj.ro/ocle/ by Dan Chiorean.
[11] http://www.kermeta.org/ by Jean-Marie Mottu.
[12] http://www.fots.ua.ac.be/events/ocl2008/

Shortcomings of the Embedding of OCL into QVT ImperativeOCL

Fabian Büttner and Mirco Kuhlmann

University of Bremen, Computer Science Department
{green,mk}@tzi.de

Abstract. MOF QVT introduces ImperativeOCL as an imperative language for operational descriptions of model transformations (QVT operational mappings). ImperativeOCL extends conventional OCL by expressions with side-effects. A couple of semantical problems arise from the way OCL is embedded into ImperativeOCL – imperative expressions are modelled as a subtype of OCL expressions. This paper points out these semantical problems and proposes a change to the operational mappings language of QVT that resolves these problems, following an approach that reuses OCL by composition rather than by inheritance in the abstract syntax of ImperativeOCL. The proposed change reduces the complexity of the imperative language, removes undefinedness, and leaves OCL conformant to its original definition.

1 Introduction

OCL [9] has proven to be a valuable ingredient in modeling, model validation, and model transformation. It can be used to precisely describe model constraints such as invariants, guards, and pre- and post-conditions, and to formulate queries to system states in general. In model transformation, it can be used to express queries to models, e.g., to specify source objects for transformations.

By now, several OCL tools exist, including ATL [2], the Dresden OCL toolkit [4], Eclipse MDT OCL [8], KMF [1], OCLE [3], Octopus [6], RoclET [12], and USE [5].

In MOF QVT [10] OCL is extended to so-called ImperativeOCL as part of QVT's "operational mappings". Within ImperativeOCL, statements with side-effects can be formulated. It adds facilities to manipulate system states (e.g, to create and modify objects, links, and variables) and certain constructs from imperative programming languages (e.g., loops, conditional execution). ImperativeOCL is used in QVT to specify transformations operationally (complementary to the relational language of QVT).

While the usefulness of a combination of OCL with imperative language elements is unquestioned, we criticise the way OCL is extended to ImperativeOCL. The chosen abstract syntax leads to a couple of semantical problems which we point out in this paper. In our opinion, the intention of ImperativeOCL can be achieved without modifying the semantics of OCL itself following an approach that favours a composition of OCL into an imperative language over reuse by inheritance in the abstract syntax.

This paper is structured as follows: In Sect. 2 we give a short overview of ImperativeOCL and its role in QVT. In Sect. 3 we explain the various semantical problems that arise from the QVT definition of ImperativeOCL. We suggest a change to the QVT specification that resolves these problems in Sect. 4 and conclude in Sect. 5.

2 ImperativeOCL

QVT defines two ways to express model transformations, a declarative approach and an operational approach. The declarative approach is the Relations language where transformations between models are specified as a set of relations that must hold for the transformation to be successful. The operational approach allows either to define transformations using a complete imperative approach or allows complementing relational transformations with imperative operations implementing the relations. The imperative language introduced by QVT is called ImperativeOCL.

ImperativeOCL adds imperative elements to OCL which are typically found in general purpose programming languages such as Java. Its semantics is defined in [10] as usual by an abstract syntax model. The complete abstract syntax of ImperativeOCL is depicted in Fig. 1.

An element of this extension is the ability to use block expressions to calculate a given value. The compute expression (meta-class ComputeExp in Fig. 1)

```
compute( v:T=initExpr ) { e1; ... ; en }
```

returns the value of the variable v after ending the sequential execution of the body *(e1; ... ;en)*. Within the body, variables defined in outer scopes can be freely accessed and changed.

New loop expressions such as *forEach* and *while* have been introduced to iteratively execute expressions (meta-classes ForExp and WhileExpr):

```
company.employees->forEach(c) { c.salary := c.salary * 1.1}

while(x<10) { x := x + 2 }
```

An imperative version of conditional evaluation is available (meta-class AltExp):

```
if ( x < 0 ) { x := 0 } else { x := 1 } endif
```

Variables can be declared in the current scope using the *var* statement (meta-class VariableInitExp):

```
var x : String
var y : Integer := 2
```

Instances of classes can be created using a *new* operator (meta-class InstantiationExp):

```
var p : Person
p := Person.new
```

The most important aspect of the abstract syntax to that we refer in this paper is the fact that all imperative expression classes inherit from OclExpression (at the top of Fig. 1). OclExpression is the base class for all expressions in conventional OCL (cf. [9]). Consequently, ImperativeExpressions can be used at all locations where OclExpressions occur. Thus, we can have imperative expressions consisting of OCL expressions that again consist of imperative expressions. For example,

```
var z : Set(Integer) := Set{1,2,3}->select(y |
  compute(x:Integer) { x := y * 2 } < 5
)
```

becomes a valid OCL expression under the QVT extension. The right-hand side of this imperative assignment expression is an OCL select expression. This OCL select expression requires a boolean body expression, which is given by a relational OCL expression whose left-hand side is an imperative compute expression. The body of this compute expression is an assignment expression whose right-hand side is again a conventional OCL expression (y * 2). While this is a very simple example, trickier mixtures of imperative expressions and OCL expressions exist that comprise several semantical problems. In the following section we explain these problems that all follow from the design by inheritance of ImperativeOCL.

3 Problems

This section explains semantical problems that arise from the embedding of OCL into QVT ImperativeOCL. First (and most formally) we show that ImperativeOCL redefines the interpretation of OCL expressions and that this redefined interpretation leads to undefined semantics of several OCL expressions that had a perfectly well-defined semantics under conventional OCL. Second, we show that several equivalence rules for OCL not longer hold if ImperativeOCL is around. Third, we further show that (under the current design) some of the new imperative expressions are actually redundant to conventional OCL expressions. Finally, we generalise this critique and discuss that the abstract syntax of ImperativeOCL violates the subtype substitution principle in various other locations in UML, too. All of these problems arise from the fact that the abstract syntax of ImperativeOCL allows imperative expressions at all locations where OCL expressions are expected – ImperativeExpression is modeled as a subclass of OclExpression.

3.1 Undefined Semantics for OCL Expressions

In conventional OCL, the semantics of an OCL expression e can be formally expressed by an interpretation function

$$I[\![e]\!] : \mathrm{ENV} \rightarrow \mathrm{VALUE}$$

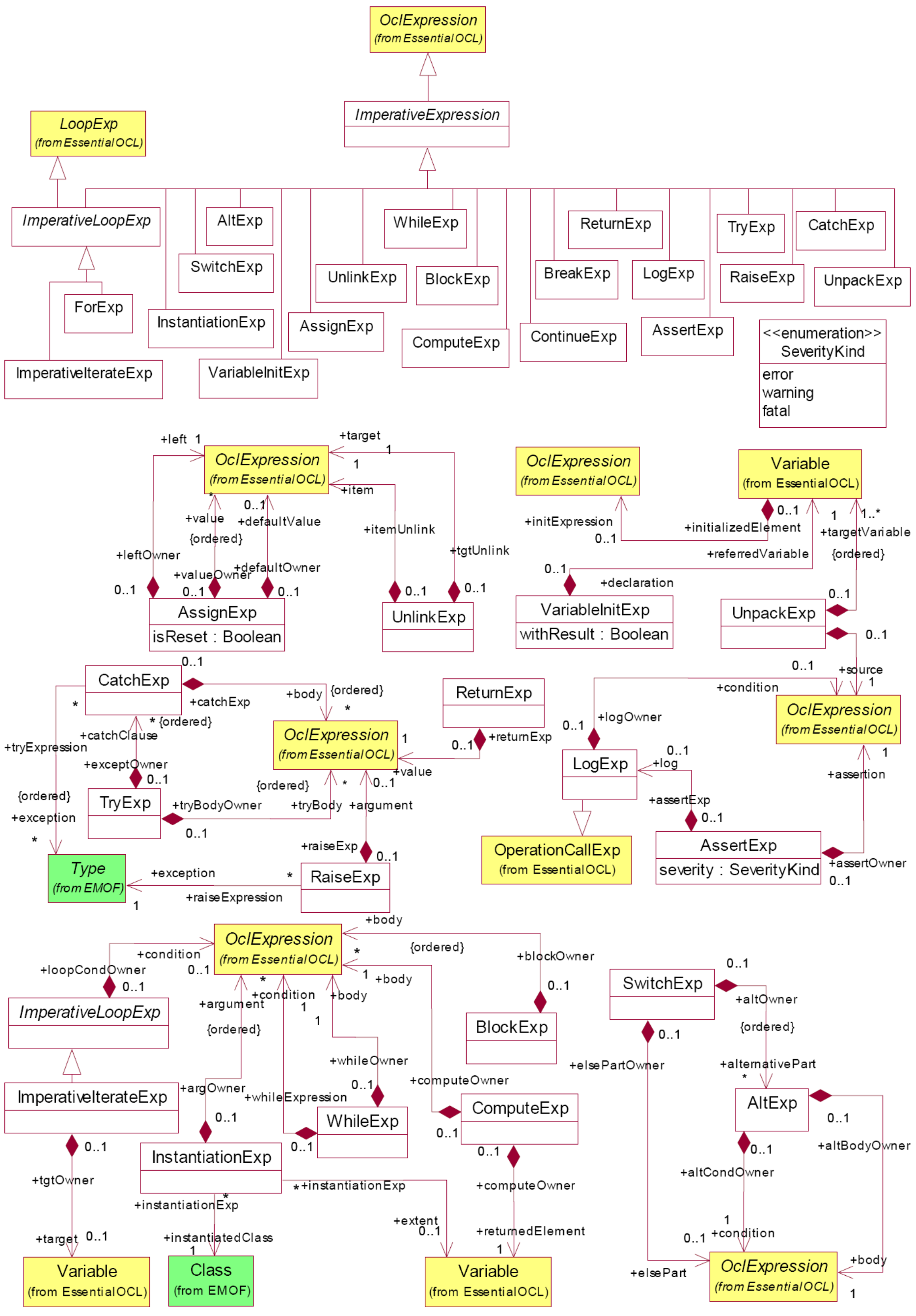


Fig. 1. ImperativeOCL abstract syntax

where ENV = (σ, β) is the environment in which the expression is evaluated (σ is a system state – objects, links, and attribute values –, and β maps bound variables to their values). Cf. [9, Annex A] and [11] for the formal semantics of OCL.

However for ImperativeOCL expressions, the interpretation function described above is not sufficient. The evaluation (or execution) of an ImperativeOCL expression does not only return a value, it also results in a (possibly) modified state and a (possibly) modified variable binding. To take this into account, the interpretation of an ImperativeOCL expression must be defined like follows:

$$I_{\mathrm{IMP}}[\![\, e \,]\!] : \mathrm{ENV} \rightarrow \mathrm{VALUE} \times \mathrm{ENV}$$

While this is not done formally in [10], the effect of all imperative expressions are described in natural language, and one could define $I_{\mathrm{IMP}}[\![\, e \,]\!]$ for all imperative expressions from this descriptions.

But, since ImperativeExpression is modeled as a subclass of OclExpression, the imperative semantics MUST be defined for all "ordinary" OCL expression, too, as ImperativeOCL expression can occur everywhere a OCL expression is expected. Figure 2 depicts this structural problem using the OCL metamodel – the redefinition of eval() in ImperativeExpression is invalid.

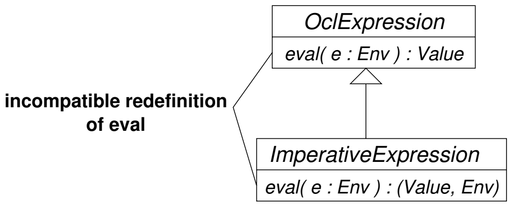

OclExpression
eval(e : Env) : Value
incompatible redefinition of eval
ImperativeExpression
eval(e : Env) : (Value, Env)

Fig. 2. Problem illustrated using the OCL meta-model

The following imperative OCL expressions illustrates this problem:

```
compute(z:Boolean) {
  var x : Boolean := true
  var y : Boolean := true
  if  ((x:=false) and (y:=false)) { ... }
  z := x
}
```

The value of this compute expression is false: the block is executed and the value of the block variable z at the end of the block becomes the value of the compute expression. Since false is assigned to x in the condition of the if statement, z becomes false at the end (the assignment expression has the value of its right-hand side, like in C or Java, therefore the expression x:=false is false).

But what happens if we change the last line as follows:

```
compute(z:Boolean) {
  var x : Boolean := true
```

```
    var y : Boolean := true
    if ((x:=false) and (y:=false)) { ... }
    z := y
  }
```

Is the value of this expression true or false? It depends on how we define the imperative semantics of the logical connectives. Given boolean expressions e_1 and e_2, we have at least two choices to define $I_{\mathrm{IMP}}[\![\, e_1 \text{ and } e_2 \,]\!](\mathrm{env})$:

1. Lazy evaluation semantics like in Java or C (returns true for the above example):

$$I_{\mathrm{IMP}}[\![\, e_1 \text{ and } e_2 \,]\!](\mathrm{env}) = \begin{cases} I_{\mathrm{IMP}}[\![\, e_2 \,]\!](\mathrm{env}_1) & \text{if } v_1 = \mathrm{true} \\ (v_1, \mathrm{env}_1) & \text{otherwise} \end{cases}$$

   where $(v_1, \mathrm{env}_1) = I_{\mathrm{IMP}}[\![\, e_1 \,]\!](\mathrm{env})$. Under this semantics (also called short-circuit evaluation) the right-hand side of the *and* operator is not evaluated if the left-hand side already evaluates to *false*. Therefore, y stays *true*.
2. Strict evaluation semantics (returns false for the above example):

$$I_{\mathrm{IMP}}[\![\, e_1 \text{ and } e_2 \,]\!](\mathrm{env}) = \begin{cases} (\mathrm{true}, \mathrm{env}_2) & \text{if } v_1 = \mathrm{true} \wedge v_2 = \mathrm{true} \\ (\mathrm{false}, \mathrm{env}_2) & \text{otherwise} \end{cases}$$

   where $(v_1, \mathrm{env}_1) = I_{\mathrm{IMP}}[\![\, e_1 \,]\!](\mathrm{env})$ and $(v_2, \mathrm{env}_2) = I_{\mathrm{IMP}}[\![\, e_2 \,]\!](\mathrm{env}_1)$. Under this semantics, both sides of the *and* operator are always evaluated. Therefore, *false* is assigned to y.

The QVT specification does not say which semantics should hold. But since ImperativeOCL expressions can occur everywhere OCL expressions can occur, this semantics has to be defined.

One can find further similar locations where the imperative semantics of OCL expression is not obvious, e.g. for the collection operation iterate (what happens if the body of the expressions modifies the range variable?) or the treatment of undefined values in arithmetic expressions (similar to the logical connectives – lazy or not?).

3.2 Breaking Equivalence Rules

In conventional OCL, several equivalence rules hold, most of them well-known from predicate logic. If we include imperative expressions into the set of OCL expressions, they all do not longer hold. This is not necessarily a problem but at least contrary to the logical character of conventional OCL.

1. Substituting variables by let expressions. In conventional OCL, the following equivalence holds:

$$\text{let } x : T = e_1 \text{ in } e_2 \Leftrightarrow e_2\{x/e_1\}$$

   In ImperativeOCL, this equivalence does not hold. The left-hand and right-hand term are only equivalent if x occurs exactly once in e2.

2. Commutativity laws.

   $$e_1 \text{ and } e_2 \Leftrightarrow e_2 \text{ and } e_1$$

   In ImperativeOCL, the commutativity laws for conjunction (and also disjunction) do not longer hold. Notice that this is a different problem than the one discussed in subsection 3.1. The following example illustrates it (returning false and true):

   ```
   compute(z:Boolean) { y := (z:=true) and (z:=false) }
   ```

   ```
   compute(z:Boolean) { y := (z:=false) and (z:=true) }
   ```

3.3 Redundancy of Existing OCL Language Features

Some of the new language features in ImperativeOCL such as forEach and the imperative conditional are not really necessary (as long as ImperativeExpression is a subclass of OclExpression). Their effect can be achieved using conventional OCL expressions:

```
company.employees->forEach(c) { c.salary := c.salary * 1.1}
```

has the same effect as

```
company.employees->iterate(c; r:OclAny=Undefined |
  c.salary := c.salary * 1.1
)
```

and

```
if ( x < 0 ) { x := 0 } else { x := 1 } endif
```

is the same as

```
if x < 0 then x := 0 else x := 1 endif
```

3.4 Further Problems

Apart from the problems illustrated above, we can find several other locations where allowing imperative expressions does not make sense. For example, ImperativeOCL would allow us to modify the system state in an invariant or a post-condition. The evaluation of the invariant could evaluate to true and invalidate the state at the same time.

Apart from the structural problems discussed above, we have the opinion that the subtype relation between ImperativeExpression and OclExpression violates the substitutability of subtypes for supertypes (e.g., [7]) very clearly. An imperative expression cannot be used everywhere a OCL expression is expected. On the contrary, there are only very few locations where imperative expressions can be safely used where an OCL expression is expected. Therefore, we propose a change to the QVT specification that leaves the semantics of conventional OCL unchanged and reuses OCL (as is) as a *part* of QVT's imperative language instead.

4 Suggested Change to the QVT Specification

We think that ImperativeOCL expressions have not been intended to be used at all locations where OCL expressions occur. Therefore, imperative languages such as the one defined in QVT (called ImperativeOCL at the moment) should use OCL by composition rather than by inheritance, as depicted in Fig. 3.

Several concrete changes in the abstract syntax of ImperativeOCL follow from this modification. Most important, two versions of assignment expressions will be required: one whose right-hand side is of type OclExpression (as current) and one whose right-hand side is an ImperativeExpression (to capture the result of a compute or instantiation expression). Figure 4 shows the modifications to the abstract syntax class diagram.

The body of imperative loops will not longer be inherited from the OCL meta-class LoopExp. Iterators and the (imperative) body expression are modeled explicitly now (Fig. 5).

Similar changes have to be made for conditional execution (meta-classes AltExp and SwitchExp), while loops (meta-class WhileExp), and general block expression (meta-class BlockExp).

The sketched modification makes a clear distinction between imperative and logical language elements (i.e., conventional OCL). This solves all of the aforementioned problems: For the OCL part (then unchanged from [9]), no underdefinedness is introduced and the expected equivalence rules hold again. Also, it is made clear that no expressions with side-effects can occur at unexpected locations such as invariants and post-conditions. Imperative loops and conditional execution are clearly separated from the logical versions.

However, constellations such as the one provided in the introduction of this paper are not longer possible if the abstract syntax of ImperativeOCL is changed

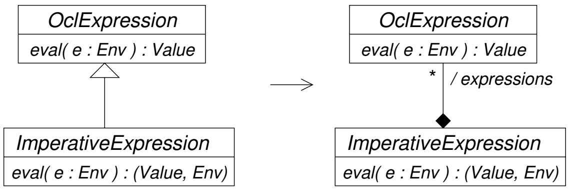


Fig. 3. Suggested change to the abstract syntax of QVT

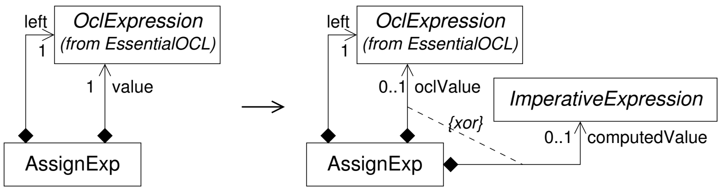


Fig. 4. Suggested change to meta-class AssignExp

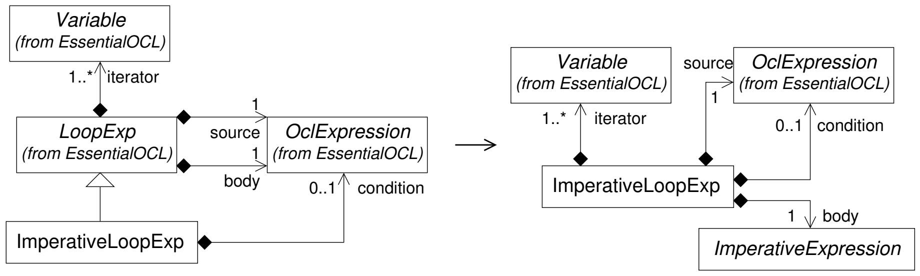


Fig. 5. Suggested change to meta-class ImperativeLoopExp

this way. While imperative expressions can still contain OCL expressions, OCL expressions can no longer contain imperative parts to calculate sub-results:

```
z := Set{1,2,3}->select(y |
  compute(x:Integer) { x := y * 2 } < 5
)
```

will be no valid expression. The example would have to be reworked either as pure OCL for the right-hand side of the assignment

```
z := Set{1,2,3}->select(y | y * 2 < 5)
```

or into a fully imperative version (except the arithmetic and relation expressions that are OCL):

```
z := Set{}
Set{1,2,3}->forEach(y) {
  if (compute(x:Integer) { x := y * 2 } < 5) { z += y }
}
```

Of course, this is a very simplified example. Imperative expressions in real QVT applications may be more complicated to rewrite. Especially, if imperative operations are invoked as part of an expression. For example the following imperative expression, using an operation with side-effects (calcAgeImperatively)

```
if (calcAgeImperatively(p1) > calcAgeImperatively(p2)) {...}
```

would have to be rewritten as

```
var ageOfP1 : Integer = calcAgeImperatively(p1);
var ageOfP2 : Integer = calcAgeImperatively(p2);
if (ageOfP1 > ageOfP2) { ... }
```

because the arguments of the relational OCL expression cannot be imperative expressions.

5 Conclusion

In this paper, we have pointed out a couple of semantical problems that all arise from the way OCL is embedded into QVT's ImperativeOCL. The design which subclasses OclExpression in the abstract syntax does not allow to replace subtype instances for supertype instances. It also requires an extended semantics of all conventional OCL expressions which is not defined at the moment.

We outlined a change to ImperativeOCL that resolves these problems by reusing OCL (as it is) in a non-intrusive way, making OCL a part of the imperative language. While this change requires certain (intermixed) expressions to be rewritten, it essentially reduces the complexity of the imperative language, removes undefinedness, and leaves OCL conformant to its original definition.

We have informed the OMG about the problems depicted in this paper by means of an OMG issue regarding [10].

References

1. Akehurst, D., Patrascoiu, O.: The Kent Modeling Framework (KMF). University of Kent (2005), `http://www.cs.kent.ac.uk/projects/ocl`
2. Allilaire, F., Bézivin, J., Jouault, F., Kurtev, I.: Atl - eclipse support for model transformation. In: Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France (2006)
3. Dan Chiorean and OCLE-Team. Object Constraint Language Environment 2.0. (2008), `http://lci.cs.ubbcluj.ro/ocle/`
4. Dresden-OCL-Team. Dresden OCL Toolkit (2008), `http://dresden-ocl.sourceforge.net/`
5. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Speci location Environment for Validating UML and OCL. Science of Computer Programming 69, 27–34 (2007)
6. Klasse Objecten. The Klasse Objecten OCL Checker Octopus. Klasse Objecten (2005), `www.klasse.nl/english/research/octopus-intro.html`
7. Liskov, B.H., Wing, J.M.: A behavioral notion of subtyping. ACM Transactions on Programming Languages and Systems 16(6), 1811–1841 (1994)
8. MDT-OCL-Team. MDT OCL (2008), `http://www.eclipse.org/modeling/mdt/?project=ocl`
9. Object Modeling Group. Object Constraint Language Specifica- tion, version 2.0, OMG document formal/2006-05-01 (June 2006), `http://www.omg.org/cgi-bin/doc?formal/2006-05-01`
10. Object Modeling Group. Meta Object Facility (MOF) 2.0 Query/View/- Transformation Specification, OMG document formal/08-04-03 (2008), `http://www.omg.org/spec/MOF/2.0/PDF/`
11. Richters, M.: A Precise Approach to Validating UML Models and OCL Constraints. PhD thesis, Universität Bremen, Fachbereich Mathematik und Informatik, Logos Verlag, Berlin, BISS Monographs, No. 14 (2002)
12. RoclET-Team. Welcome to RoclET (2008), `http://www.roclet.org/`

Optimization Patterns for OCL-Based Model Transformations

Jesús Sánchez Cuadrado[1], Frédéric Jouault[2], Jesús García Molina[1], and Jean Bézivin[2]

[1] Universidad de Murcia
{jesusc,jmolina}@um.es
[2] AtlanMod team, INRIA & EMN
{jean.bezivin,frederic.jouault}@inria.fr

Abstract. Writing queries and navigation expressions in OCL is an important part of the task of developing a model transformation definition. When such queries are complex and the size of the models is significant, performance issues cannot be neglected.

In this paper we present five patterns intended to optimize the performance of model transformations when OCL queries are involved. For each pattern we will give an example as well as several implementation alternatives. Experimental data gathered by running benchmarks is also shown to compare the alternatives.

1 Introduction

Rule-based model transformation languages usually rely on query or navigation languages for traversing the source models to feed transformation rules (e.g., checking a rule filter) with the required model elements. The Object Constraint Language (OCL) is the most common language for this task, and it is implemented in several languages such as: ATL [4], QVT [10], and Epsilon [5].

In complex transformation definitions a significant part of the transformation logic is devoted to model navigation. Thus, most of the complexity is typically related to OCL. From a performance point of view, writing OCL navigation expressions in an efficient way (e.g., avoiding bottlenecks) is therefore essential to transformations optimization.

We are currently working on the identification of common transformation problems related to performance. For each identified issue, we analyze several alternative solutions. In this work, we present some of our initial results in the form of idioms.

In particular, we describe five performance-related patterns in model transformations when OCL queries are involved. Each pattern is presented in three parts: i) a statement describing the problem, as well as a motivating example; ii) some experimental data gathered by running benchmarks, so that different implementation alternatives can be compared, and finally iii) some recommendations on the basis of this data.

The paper is organized as follows. Next section describes the five patterns. Section 3 presents some related work. Finally, Section 4 gives the conclusions.

2 Performance Patterns

In this section we describe five OCL patterns and analyze them in order to improve the performance of OCL navigation expressions in model transformations. We rely on the experimental data obtained by running performance benchmarks to compare the different implementation strategies[1].

We identified these patterns by working on actual transformations in which we identified one or more bottlenecks. The corresponding expressions are then re-implemented in a more efficient way. Next, whenever a pattern is identified, a small, synthetic benchmark is created to isolate this particular problem and to compare several implementation options.

We will illustrate the patterns using the ATL language[2], but they are general for any rule-based transformation language using OCL, such as QVT [10]. Therefore, these optimization patterns can be considered as idioms or code patterns [3], since they provide recommendations about how to use OCL.

ATL is a rule-based model transformation language based on the notion of declarative rule that matches a source pattern and creates target elements according to a target pattern. It also provides imperative constructs to address those problems that cannot be completely solved in a declarative way. The navigation language of ATL is OCL. Helpers written in OCL can be attached to source metamodel types. Also, global attributes whose value is computed at the beginning of the transformation execution can be defined. The ATL virtual machine provides support for several model handlers, but in our experiments we have only considered EMF.

It should be noted that the patterns presented here suppose that no special optimization is performed by the compiler (i.e., straightforward implementation of the OCL constructs), and that all optimizations have to be done manually by the developer. This is the case with ATL 2006, but this may not be the case with all OCL implementations. These patterns could also probably be used by an OCL compiler to perform internal optimizations using expression rewriting, but this is out of the scope of this paper.

2.1 Short-Circuit Boolean Expressions Evaluation

Model transformations usually involve traversing a source model by means of a query generally containing boolean expressions. When such boolean expressions are complex and the source model is large, the order in which such operands are evaluated may have a strong impact on the performance if short-circuit evaluation [2] is used. Short-circuit evaluation means that the first condition is always evaluated but the second one is only evaluated if the first argument does not suffice to determine the value of the expression.

[1] The benchmarks have been executed in a machine with the following configuration: Intel Pentium Centrino 1.5Ghz, 1GB RAM. Java version 1.6.0 under Linux kernel 2.6.15.

[2] The ATL version used is: ATL 2006 compiler, using the EMFVM virtual machine on Eclipse 3.4.

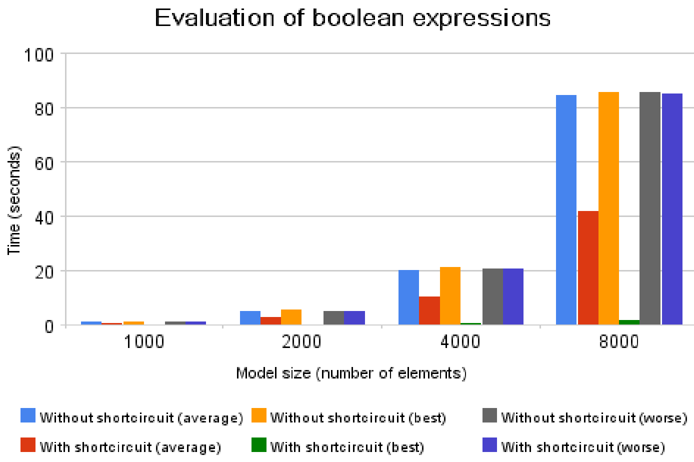


Fig. 1. Boolean expressions with and without short-circuit-evaluation

Experiments. The experiment carried out compares the possible performance impact of evaluating boolean expressions with and without short-circuit evaluation.

Figure 1 shows the execution time of a query containing a boolean expression with the following form: `simpleCondition or complexCondition`, where *simpleCondition* is an operation with a constant execution time, while *complexCondition* is an operation whose execution time depends on the size of the source model. The query is executed once for each element of a given type in the model.

Three cases has been considered in the benchmark, each one tested with and without short-circuit evaluation:

– The *simpleCondition* operation returns true for half of the elements. This means that *complexCondition* must be executed, in any case, for the other half of the elements. This can be considered an average case.
– In the second case, *simpleCondition* is satisfied for all the elements. Thus, *complexCondition* may not be evaluated. This would be the best case.
– In the third case, *simpleCondition* is not satisfied for any element. Thus, *complexCondition* must always be evaluated. This would be the worst case.

As expected, in the average case the performance improvement with short-circuit evaluation is directly proportional to the number of times the first condition is executed (i.e. its execution prevents the execution of the second one). In the best case the execution time with short-circuit evaluation is much lower because the complex condition is never executed. On the contrary, for the worst case the complex condition must always be executed, so there is no difference between having short-circuit evaluation or not.

Recommendations. There are two well-known strategies to improve the performance of a boolean expression when short-circuit evaluation is considered, depending on whether it is "and" or "or".

- **And**. The most restrictive (and fastest) conditions must be placed first, that is, those conditions/queries more likely to return a "false" value. Thus, the slowest condition will be executed less often.
- **Or**. The less restrictive conditions (and fastest) must be placed first, that is, those conditions/queries more likely to return a "true" value. Again, the slowest condition will be executed less often.

If the implementation supports short-circuit evaluation then boolean expressions can be written in such a way that efficiency is considerably improved. If not, any expression can be rewritten using the rules of the following table, where the second column shows how to rewrite the expressions in OCL. It is important to notice that this table assumes that the results of the queries are defined values, i.e. true or false, but not OclUndefined. As a matter of fact, the current implementation of ATL uses a two-valued logic.

	With short-circuit	Without short-circuit
AND	`query1() and query2()`	`if query1() then query2() else false endif`
OR	`query1() or query2()`	`if query1() then true else query2() endif`

2.2 Determining an Opposite Relationship

Given a relationship from one model element to another, it is often necessary to navigate through the opposite relationship. For instance, if one is dealing with a tree defined by the *children* relationship it may be necessary to get a node's parent node (i.e., navigating the opposite relationship of *children*).

If the opposite relationship has been defined in the metamodel, then navigation in both directions can be efficiently achieved. However, such opposite relationship is not always available, so an algorithm to check all the possible opposite elements has to be worked out. This algorithm tends to be inefficient since it implies traversing all the instances of the opposite relationship's metaclass.

When the metametamodel supports *containment* relationships, and the reference we are considering has been defined as *containment*, then it is possible for a transformation language to take advantage of the unique relationship between an element and its container to efficiently compute the opposite one. For instance, ATL provides the `refImmediateComposite()` operation (defined in MOF 1.4) to get the container element.

Experiments. The performance test has consisted in getting the owning package for all classifiers of a given class diagram. A package references its owned classifiers through a `classifiers` relationship. A helper for the `Classifier` metaclass has been created to compute the `owner` opposite relationship. The helper will be called once for each classifier of the model.

Four ways of computing an opposite relationship have been compared:

– Using an iterative algorithm such as the following (all examples are given in ATL syntax[3]):

```
helper context CD!Class def : parent : CD!Package =
    CD!Package.allInstances()->any(p |
       p.classifiers->includes(self) );
```

– Using the *refImmediateComposite()* operation provided by ATL.
– Precomputing, before starting the transformation, a map (dictionary in QVT terminology) associating elements with their parent. In the case of ATL, maps are immutable data structures, and as we will see this issue affects performance.

```
helper def : pkgMap : Map(CD!Class, CD!Package) =
  CD!Package.allInstances()->iterate(p;
                    acc  : Map(CD!Class,CD!Package) = Map{} |
    p.classifiers->iterate(c;
                    acc2 : Map(CD!Class, CD!Package) = acc |
      acc2.including(c, p)
    )
  );

helper context CD!Class def : owner : CD!Package =
  thisModule.pkgMap.get(self);
```

– The same as the previous strategy but using a special mutable operation to add elements to the map.

The results of this benchmark are shown in Figure 2. Three class diagrams with $n \times m$ elements, where n is the number of packages and m is the number of classes per package, have been considered at this time: (1) a small model with 10 packages and 250 classes per package, (2) a second model with 25 packages and 500 classes per package, and (3) a third model with 500 packages and 25 classes per package. This last model has been introduced to test how the "shape" of the model may affect the performance.

The `refImmediateComposite` operation proves to be the best option, however the performance of the "mutable map version" is comparable. The iterative algorithm is more efficient than the "immutable map version" when the number of packages is smaller than the number of classes, which will be probably the common case. The reason is that such an algorithm only iterates over the packages, while the "map version" also iterates over all the classes. The main reason for the poor numbers is that, since it is immutable, the cost of copying a map each time a new element is added is too high.

[3] The *helper* keyword and the terminal semicolon are required by ATL to syntactically identify OCL helpers. ATL also requires that type names be prefixed by the name of the metamodel defining them (e.g., *CD* here), separated by an exclamation mark.

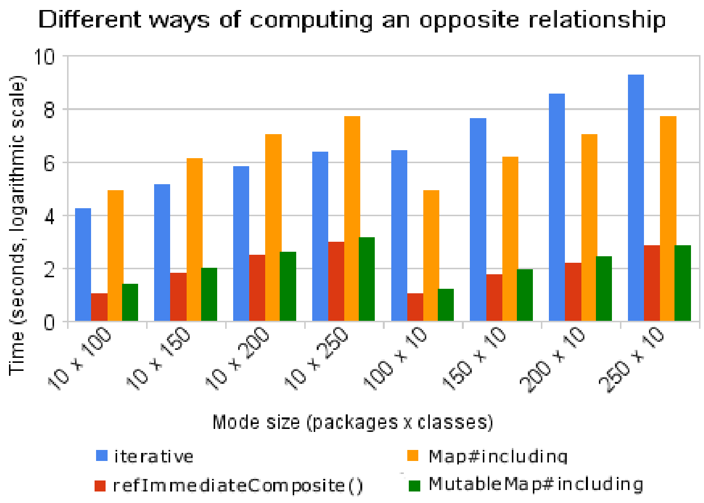


Fig. 2. Comparison of different ways of computing an opposite relationship. The logarithmic scale used for the time axis corresponds to the following formula: $1.59 \times ln(time) + 8.46$.

Recommendations. According to this data, to compute the opposite of a containment relationship the `refImmediateComposite` operation should be used. If the reference is not containment or just the metametamodel does not provides this feature, using a mutable map proved to be the best option.

If the transformation language does not provide a mutable map data type, but an immutable one, the iterative algorithm or the map strategy has to be chosen according to the most usual shape of the models.

As a final remark, although maps or dictionaries are not natively supported by OCL, transformation languages usually extends OCL to implement them. For instance, ATL provides an immutable Map[4] data type (usable in side-effect free OCL expressions), and QVT provides a mutable Dictionary data type.

2.3 Collections

OCL provides different collection data types. Each type is more efficient for certain operations and less efficient for others. It is important to choose the proper collection data type according to the operations to be applied, otherwise performance may be compromised.

In this section we compare the implementation of the `including` (adds an element to a collection), and `includes` (check the existence of some element) operations for three collection data types, `Sequence`, `Set`, and `OrderedSet`. The performance results are applicable to other operations, such as `union`.

[4] In order to measure the performance using a mutable Map, we had to implement it in a test version of the ATL engine.

Experiments. The benchmark for the `including` operation consists of iterating over a list of n elements, adding the current element to another list in each iteration step. The execution time for different input sizes, as well as for the three collections data types is shown in Figure 3.

As can be seen, `including` is more efficient for sequences than for sets. This is what one would expect, since in a sequence a new element is inserted at the tail, and there is no need to check if the element was already added. The performance of ordered sets is slightly worse than sets, basically because it is internally implemented in ATL using a `LinkedHashSet`, that is, both a hash map and a linked list must be updated in each insertion.

The benchmark for the `includes` operation consists of finding an element which is in the middle of a list of n elements. The same code is executed 2000

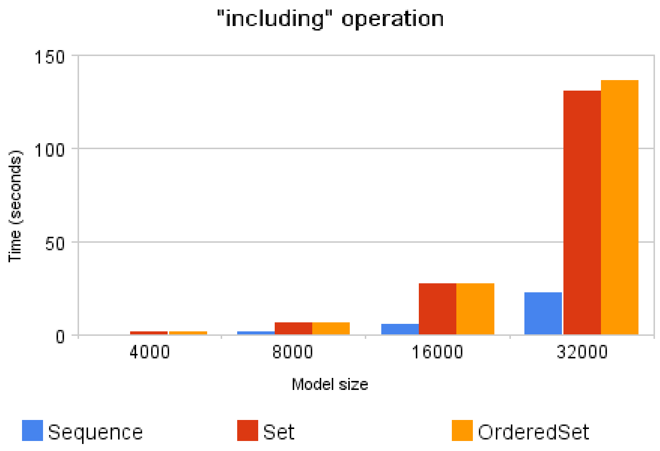


Fig. 3. Comparison of the `including` operation for different collection data types

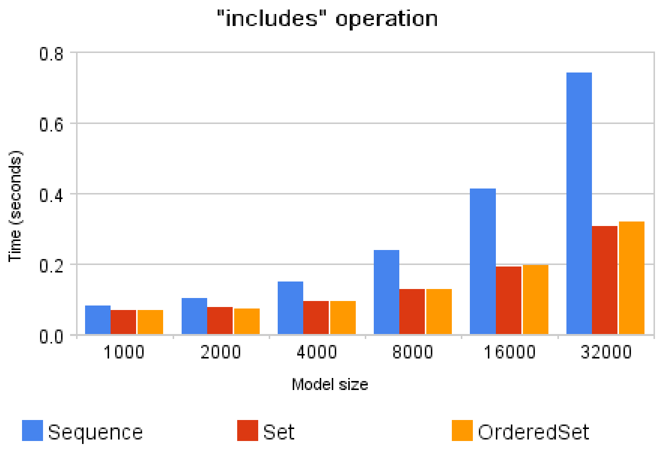


Fig. 4. Comparison of the `includes` operation for different collection data types

times. The execution time for different input sizes, as well as for the three collections data types is shown in Figure 4.

As expected the cost of `includes` is greater for sequences. However, if it executed less times, for instance 100 times, the execution time is similar in all cases, and there is not difference in using a sequence or a set. This shows that it is not worth converting a collection to a set (using `asSet`) if the number of query operations (such as `includes`) is not large.

Recommendations. The decision about which collection data type to use should be based on which will be the most frequent operations. In particular, these tests show that unless one needs to make sure that there is no duplicated elements into the collection (or if the transformation logic cannot enforce it), then the sequence type should be used, in particular when operations to add new elements are frequently called.

2.4 Usage of Iterators

OCL encourages a "functional" style of navigating through a model by promoting iterators to deal with collections. Thus, queries are often written without taking into account the efficiency of the expressions, but just trying to find out a readable, easy or more obvious solution.

For instance, it is common to come across OCL code like expression (a) shown below, which obtains the first element of a collection satisfying a condition. However, expression (b) may be more efficient since the iteration can finish as soon as the condition is satisfied. Of course, an optimizing compiler could rewrite (a) into (b).

```
(a) collection->select(e | condition)->first()
(b) collection->any(e | condition)
```

Thus, it is important to take into account the contract of each iterator and operation to choose the most appropriate one, depending on the computation to be performed.

Experiments. To assess whether it is really important, from a performance point of view, to be careful when choosing an iterator we have compared these two ways of finding the first element satisfying a condition in a list of n elements. In this benchmark the condition is satisfied by all elements after the middle of the list. This means that option (a) will return a list of $n/2$ elements.

The first time we ran this benchmark, the execution time for both cases (a) and (b) was the same. The reason is that the current implementation of `any` in ATL does not finish the iteration as soon as possible, but it is equivalent to "select()->first". Thus, a new optimized version was implemented and its performance is also compared.

Figure 5 shows the execution time for the three cases: using the original version of `any`, using a fixed version and with "select()->first". It also shows another case which is explained below.

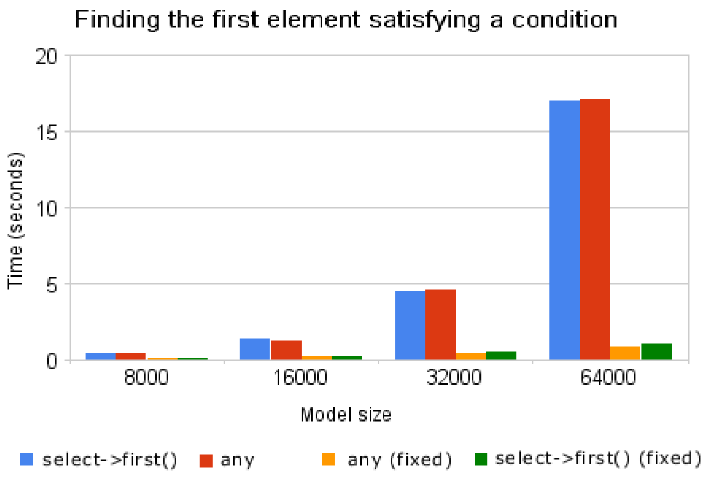


Fig. 5. Finding the first element satisfying a condition

According to the proposed benchmark, the execution time of "select()->first" should be worse than using `any`, but not so much. We looked into this issue and the reason is related to the implementation of the `select` operation in ATL. It internally uses the standard OCL `including` operation to add an element to the result each time the condition is satisfied. Since `including` is an immutable operation the whole partial result is copied for each selected element. That is why as the size of the list grows the execution time grows exponentially.

We implemented an optimized version of `select` which uses a mutable operation to add elements to the result. As can be seen in Figure 5, its performance is greater than the original, but it is also comparable to `any`. The main reason for this result is that the transformation execution involves a constant time which is independent of the iterator execution time. When such constant time is removed, the `any` iterator is around 150% faster.

Recommendations. The `select` iterator should be used only when it is strictly needed to visit all elements of the collections. Iterators such as *any*, *exists*, *includes*, etc. should be used to avoid iterating the whole collection. In any case, the benchmark results show that if the `select` iterator is properly implemented then it can provide a performance comparable to other iterators.

2.5 Finding Constant Expressions

In rule-based transformation languages, rules are executed at least once for each instance matched against the source pattern, so all expressions within the rule may be executed once for each rule application. When such expressions depend on the rule's source element, then it is inevitable to execute them each time. However, those parts of an expression which are independent of variables bound to the source element can be factorized in a "constant", so that they are executed only once when the transformation starts. Some transformation engines do not support this kind of optimization, so it has to be done manually.

As an example, let us consider the following transformation rule, which transforms a classifier into a graphical node. The condition to apply the rule is that the classifier (instance of `Classifier`) must be referenced by another element which establishes whether or not it is drawable (`Drawable`). Since the filter is checked for each classifier, all elements of type `Drawable` are traversed each time the engine tries to find a match.

```
rule Classifier2Node {
  from source : CD!Classifier (
     DrawModel!Drawable.allInstances()->exists(e | e.element = source)
  )
  to Graphic!GraphNode ...
}
```

A more efficient strategy is to compute in a constant attribute all the drawable elements, so that the transformation can be rewritten in the following way:

```
helper def : drawableElements : Set(CD!Classifier) =
  CD!Drawable.allInstances()->collect(e | e.element);

rule Classifier2Node {
  from source : CD!Classifier (
    thisModule.drawableElements->includes(source)
  ) ...
}
```

Experiments. The rule shown above has been executed for several input models (the same number of elements of type `Classifier` and `Drawable` is assumed). Figure 6 shows the results for the following three cases: (1) without

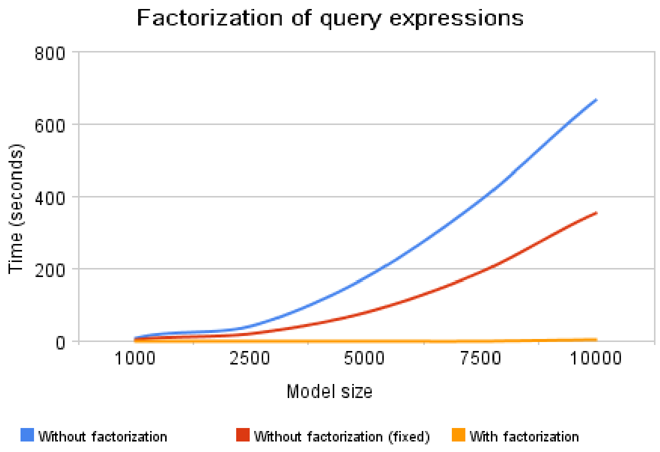


Fig. 6. Performance impact of the factorization a common expression into a constant

pre-computing the common query code into a constant, and using the original ATL version of `exists`, (2) the same but using an optimized version of `exists` which finishes the iteration as soon as it finds the required element, and (3) using the strategy of pre-computing a constant.

As can be seen the third strategy has a cost which is significantly lower than the two others, and does not grow as fast (the algorithm complexity is $O(n+m)$), while the first one has a cost of $O(n \cdot m)$, where n is the number of elements of type `Classifier` and m is the number of elements of type `Drawable`.

Recommendations. It is possible to easily identify expressions within rules and helpers which can be factorized into some constant because they usually rely on `allInstances()` to get model elements without navigating from the rule's source element. Therefore, the transformation developer should be aware of this kind of optimization and apply it whenever possible.

Also, it is worth noting that using a `let` statement (or similar) is a good practice to factorize expressions at the local level.

3 Related Work

In [6] the need for developing benchmarks to compare different OCL engines is mentioned. The authors have developed several benchmarks that can be found in [1]. However, they are intended to compare features of OCL engines, rather than performance.

In [8] and [9] the authors present several algorithms to optimize the compilation of OCL expressions. They argue that its optimizing OCL compiler for the VMTS tool can improve the performance of a validation process by 10-12%.

Regarding performance of model transformation languages few work has been been done. In [11] a benchmark to compare graph-based transformation languages is proposed. In [7] some general recommendations about how to improve performance of model driven development tools are presented, but neither concrete examples or experimental data are given.

4 Conclusions and Future Work

In this paper we have presented several optimization patterns for OCL-based transformation languages. These patterns address common navigation problems in model transformations from a performance point of view. For each pattern we have provided several implementation options along with performance comparison data gathered from running benchmarks.

The contribution of this work is twofold, on the one hand these patterns may be useful as a reference for model transformation programmers to choose between different implementation alternatives. On the other hand, they provide some empirical data which is valuable for tool implementors to focus on the optimization of some common performance problems.

As future works we will continue defining benchmarks for model transformations in order to identify more patterns related to performance. We are also improving our framework for benchmarking to consider other transformation languages. Beyond the individual patterns that are being identified, we are also looking at improving and generalizing a method for finding, identifying and classifying transformation patterns.

References

1. OCL benchmarks, `http://muse.informatik.uni-bremen.de/wiki/index.php/ocl_benchmark_-_core`
2. Aho, A.V., Ullman, J.D.: Principles of Compiler Design. Addison-Wesley series in computer science and information processing. Addison-Wesley, Reading (1977)
3. Beck, K.: Implementation Patterns. Addison-Wesley Professional, Reading (2006)
4. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
5. Kolovos, D.S., Paige, R.F., Polack, F.A.: The epsilon transformation language. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 46–60. Springer, Heidelberg (2008)
6. Kuhlmann, M., Gogolla, M.: Analyzing semantic properties of ocl operations by uncovering interoperational relationships. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735. Springer, Heidelberg (2007)
7. Langlois, B., Exertier, D., Bonnet, S.: Performance improvement of mdd tools. In: EDOCW 2006: Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops, p. 19. IEEE Computer Society, Los Alamitos (2006)
8. Mezei, G., Levendovszky, T., Charaf, H.: Restrictions for ocl constraint optimization algorithms. Technical report, Technische Universität Dresden (October 2006)
9. Mezei, G., Levendovszky, T., Charaf, H.: An optimizing ocl compiler for metamodeling and model transformation environments. In: Software Engineering Techniques: Design for Quality, pp. 61–71. Springer, Heidelberg (2007)
10. OMG. Final adopted specification for MOF 2.0 Query/View/Transformation (2005), `www.omg.org/docs/ptc/05-11-01.pdf`
11. Varro, G., Schurr, A., Varro, D.: Benchmarking for graph transformation. In: VL-HCC 2005: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 79–88. IEEE Computer Society, Los Alamitos (2005)

Third International Workshop on Quality in Modeling

Jean-Louis Sourrouille[1,2], Ludwik Kuzniarz[3], Lars Pareto[4], Parastoo Mohagheghi[4], and Miroslaw Staron[5]

[1] Univ Lyon
[2] INSA-Lyon, LIESP, F-69621, Villeurbanne, France
[3] Blekinge Institute of Technology, Ronneby, Sweden
[4] SINTEF ICT, Oslo, Norway
[5] IT University of Göteborg, Göteborg, Sweden

Abstract. Software quality management is widely researched within Model Driven Software Development (MDD), from both industry practices and academic research viewpoints. The goal of this workshop was to gather researchers and practitioners interested in the emerging issues of quality in the context of MDD. During the first part of the workshop, selected papers were presented and discussed. The second part was divided into two working sessions. The first session was devoted to the introduction of model quality into the software development process by drawing a parallel with quality of code. An invited practitioner introduced issues related to quality of code, followed by a guided discussion based on a list of predefined questions. The second session was dealing with future work and research interests of the participants.

1 Introduction

Quality is an important issue in software engineering, and stakeholders involved in the development of software systems definitely are aware of the impact of the quality of both development process and produced artifacts on the quality of final system. The recent introduction of Model Driven Software Development (MDD) raises new challenges related to ensuring proper quality of the software produced when using this approach. Software quality management within MDD is widely researched from multiple perspectives. Furthermore, in software engineering, the issues of model quality need to be approached from the viewpoints of both industry practices and academic research in order to arrive at sound and industrially applicable results.

The Quality in Modeling series of workshop (QiM) aim to provide a forum for presenting and discussing emerging issues related to software quality in MDD. The intended result is to increase consensus in understanding quality of models and issues that influence this quality. In the previous QiM workshop, a common quality model was established [1]. The intention of this year's workshop was to discuss model quality issues related to software development processes. Within "usual" software development, code quality seems to be under-exploited. However, all the concepts and theory about code quality have been widely described. Therefore, a special attention is to be paid to practical issues such as the introduction of model quality into the software development process in a convenient and accepted way.

2 Summary of the Paper Contributions

The presentation of papers consisted of two sessions, each one with three topics addressed by the papers [4]:

- Towards model quality
  - Definition of a measurement procedure to accurately quantify the size of software developed with a Model-Driven Development (MDD) [4],
  - Definition of a proactive and process-driven approach based on a meta-approach to be instantiated in every sub-process producing a UML model [5],
  - Description and implementation of a customized style guide for UML [6].
- Frameworks for model quality
  - Definition of an operational framework to address database schema quality through both global and analytical views of quality [7],
  - Empirical validation of measures for UML class diagrams through a meta-analysis study from five controlled experiments [8],
  - Definition of a metamodel to precisely define quality elements and their relationships in a quality model [9].

3 Introducing Model Quality in the Development Process

The starting point of the discussion was the introduction, presented by an industrial expert, Marc Rambert from *Kalistick*, on how code quality management is approached in an industrial context [2]. Obviously, there are common points between model quality and code quality, but the main reason for this discussion was the actual use of quality in practice. Despite the fact that theoretical and practical aspects of code quality are well-known, and that a number of market tools for assessing code quality exist for a long time, the quality of code is not used as much as it could be. To analyze deeply theoretical aspects of quality is not enough. We have certainly to deal with practical and/or human aspects for model quality to enforce its usage by software developing teams. The aim of this discussion was to take lessons from experience related to code quality to increase our chances to introduce successfully model quality into the software development process. First, the industrial expert recalled issues related to code quality:

- Getting control over the technical quality of a software development, which requires clear objectives closely linked with project management, and aid to teams to achieve project goals,
- Improving the technical quality by providing teams with a set of good practices, detecting gaps between actual code and objectives, and focusing on key issues for improvement,
- Providing a balance between additional costs and returns on investment. Quality is not an absolute value; it is related to objectives that depend on the needs of the target application. For instance the requirements for an embedded system are not the same than for a text editor.

The expert also showed examples of metrics related to code quality, how quality can be assessed and visually reported to assist teams in their daily work and taking decisions for release. After this introduction, a discussion guided by a set of questions sent to the participants before the workshop was carried on. For each of the questions the course was as follows: the question was recalled followed by the moderator's comments; then the question was answered from the code quality perspective, and finally an open discussion was carried out aiming to find a "common" answer from a model quality perspective. In the sequel, a *quality problem* means a quality level lower than the expected one for a criterion, for instance the number of dependency cycles is too large. A *quality goal* is the expected value for some aggregate of assessments and metrics. Quality goals depend on the application requirements and should be defined by stakeholders before starting the development. In the following we list the questions and a summary of the discussion on each question.

1. Not all metrics can be measured automatically. Some quality properties such as architecture design value can be assessed manually.
   Should we keep only automatic measurement?
   To what extent can we keep manual assessment?

   The overall answer is that it should be as automatic as possible depending on the required effort. If the semantics is to be considered then it should be manual. Even when the assessment is automatic, a manual interpretation is necessary in order to assess where we are and to undertake recovery actions.

2. Software quality is relative to the requirements of an application: there is no absolute level of quality, and "over-quality" is just non-quality.
   How to define/express the quality requirements of an application from the end user point of view?
   How to avoid overestimated quality requirements, which implies over-costs?

   The schema Fig. 1 expresses a basic principle: increasing the development costs to reduce maintenance costs is limited by the increase in total costs. To define the suitable quality level, the industrial expert draws a quality profile (Fig. 2) from replies to a questionnaire directly related to project requirements but only indirectly related to quality. Stakeholders have different requirements and they should reach an agreement. The quality requirements for models seem to be closer to system quality requirements than quality requirements for code, which might result in some differences in the description.

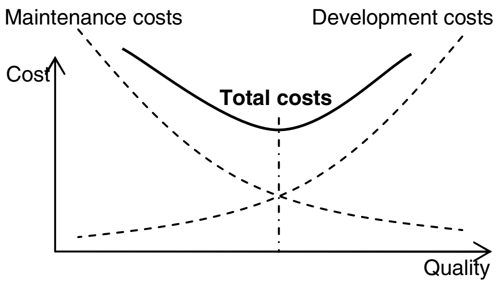


Fig. 1. Optimal cost

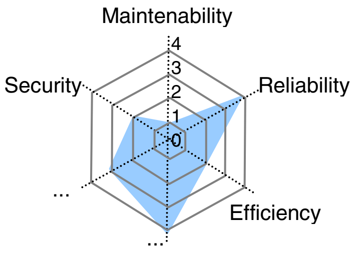


Fig. 2. Quality profile

3. When actual quality is different from the expected one, some quality goals may have a greater importance for the end user.
   How to express the importance of application's quality goals?

   In the industrial context, a dashboard shows the gap between actual and expected profiles, and improvement plans are proposed to reduce the gaps. All quality aspects have not the same importance for the stakeholders, leading to two kinds of conflicts: (i) internal, between quality attributes, for instance to increase maintainability may reduce performance; (ii) external, the same attribute being more important for one stakeholder that for another one.

4. Continuous quality assessment would be a burden for the developer and would probably result in some reject. On the other hand, to find quality problems as early as possible is a well-known need in software engineering.
   What could be the frequency or suitable moment for quality assessment?

   The frequency may depend on cost of checks. In practice, weekly assessment is enough for code. When no gap between actual values and goals is detected, and when the trend of evolution does not deviate, the human cost is quite null; otherwise a quick reaction is possible. Regarding models, instead of regular checks, quality gates or check points could be introduced in the development process to ensure quality levels.

5. When quality problems are detected, there are numerous possible actions. These actions generally depend on the application status: to correct existing software might be risky (non-regression) and of limited interest, while to correct software under construction will have a better return.
   Should we help the user to define the actions to achieve?
   Should we prioritize the quality problems to deal with?
   Finally, should the developer deal with all the quality problems?

   Another factor affects the decision: how fast the developer can find the fault? Actions to undertake depend on project management decisions, and to help managers to make their decisions is necessary.

6. There is a link between metrics and quality goals.
   To what extent does the developer need to know this relationship?
   When this relationship is known, is it acceptable for a developer to limit quality problems to stay just below thresholds that triggers quality problems?

   Example: if the metric for readability is that a sequence diagram includes less than 20 connected elements, the quality goal can specify that at most 5 sequence diagrams may break the rule. Below 5 rule violations, no quality problem is detected, therefore the developer is not warned and the situation is acceptable.

7. To find a way to deal with a quality problem is not always easy.
   Do we need to aid correcting quality problems?
   Do we need to show the place of the problem in the model (see tool below)?

   Tools detect the symptoms but not the causes: the tool may detect that there are too many dependencies, which does not explain why and how to reduce them. Similarly, patterns and anti-patterns explain why and where the problem is, but not how to correct it. Since errors may come from a combination of several causes, explanations should be precise. Regarding model quality, to help developers implies embedding quality checks in a tool, which is a great difference with code.

8. Developer training could be an important aspect to get better results.
   What kind of training would be useful: before development? When errors occur? Should developers training focus on actual metrics? Or only on principles?

   The experience shows that teaching rules has no interest. The best way seems to train on principles, and when a problem occurs to train on this problem. Moreover, training professionals has psychological issues to take into account.
9. Many stakeholders will look at quality results, each one with a different point of view.
   How to present the results in a suitable way for each stakeholder?
   What are the suitable ways for developers?

   A dashboard for each category of stakeholder is useful in industrial context.
10. Code quality assessment does not depend on a tool.
   Is it desirable to assess model quality independently of any tool (for instance by analyzing XMI files)? Or should the IDE tool include quality assessment?

   Participants mainly think that tools should include quality assessment. This ideal solution requires customizing each tool.

Comments. Due to lack of time, but also because all the answers are not actually known, there is no precise answer for each question. The list was limited to 10 items, aiming to tackle different practical aspects about model quality. Anyway, answers will be needed to bring model quality into play. These aspects complement the quality model discussed during the previous workshop.

4 Working Session on a Road Map for Further Research

The second session of the working part was devoted to development of a common research roadmap, by the following procedure. 1) The presentations were analyzed using the unified quality model of the QiM'07 workshop [10]; analysis result was a graph associating presentations to distinct qualities of concern. 2) This graph was presented for review: contributing sites were asked to check that associated qualities were appropriate and complete with respect to the research pursued at the site. 3) The graph was revised collectively. The outcome is given in the Table below. The topmost part shows the home base of the research groups behind the QiM'08 contributions. The lowermost part shows the white spots with respect to the common quality model: these qualities have no immediate connection to the presented talks, and should be fertile ground for research.

Acknowledgement. We would like to thank Marc Rambert, from Kalistick, provider of the 1^{rst} SaaS platform for code quality, who kindly agreed to prepare the invited talk for the working session, and to moderate the discussion based on code quality.

		P1 Valencia	**P2 Genova**	**P3 Lyon**	**P4 Namur**	**P5 Ciadud**	**P6 Oslo**	**P7 Ville-Urbaine**
Model Quality	Correctness		x	x			x	
	Consistency		x	x			x	
	Completeness		x				x	
	Conciseness		x				x	
	Maintainability				x	x	x	x
	Complexity					x		
	Size	x				x		
	Understandability				x	x	x	
	Transferability							x
Infrastructure	Automatic	x		x				
Process Quality	Rigourously defined	x	x	x			x	
	Automateable							x
	Easily configureable							x
Project Quality	Rigourously managed							x
	Automatically Measurable							x
	Modeling Guidelines			x				

White Spots	**Model Quality**	Navigability, Traceability, Measurable, Stable, Precision Improving,, Detailedness
	Infrastructure Quality	Ubiquitous, Updateable, Seclusive, Flexible, Categorial, Archival, Cohesive, Efficient
	Process Quality	Predictability, Reuse of good practices, Roll backing, Quality Assurance for models, Explicit about modeling purpose, Voluntary, Incentive, Regulatory, Process Support, Measurability, Effectiveness, Productivity
	Project Quality	Qualified staff**,** Skill, Experience, Tools

References

1. Kuzniarz, L., Pareto, L., Sourrouille, J.-L., Staron, M.: Third intenational workshop on quality in modeling. In: Giese, H. (ed.) MoDELS 2007 Workshops. LNCS, vol. 5002, pp. 271–274. Springer, Heidelberg (2008)
2. Blanc-dit-Grenadier, N., Rambert, M., Sourrouille, J.-L., Aubry, R.: Toward a real integration of quality in software development. In: ICSSEA 2008 (2008)
3. Sourrouille, J.-L., Staron, M., Kuzniarz, L. (eds.): Proc. of the 3rd Workshop on Quality in Modeling, IT University of Göteborg RR 2008: 02, ISSN 1654-4870, pp. 1–88
4. Maron, B., Condori-Fernandez, N., PastorJean, O.: Design of a Functional Size Measurement Procedure for a Model-Driven Software Development Method. In: [3]
5. Reggio, G., Astesiano, E., Ricca, F.: A proactive process-driven approach in the quest for high quality UML models. In: [3]
6. Hindawi, M., Morel, L., Aubry, R., Sourrouille, J.-L.: Description and Implementation of a Style Guide for UML. In: [3]
7. Lemaitre, J., Hainaut, J.-L.: A Combined Global-Analytical Quality Framework for Data Models. In: [3]
8. Manso, M.E., Cruz-Lemus, J.A., Genero, M., Piattini, M.: Empirical Validation of Measures for UML Class Diagrams: A Meta-Analysis Study. In: [3]
9. Mohagheghi, P., Dehlen, V., Neple, T.: Towards a Tool-Supported Quality Model for Model-Driven Engineering. In: [3]
10. Pareto, L., Lange, C., Mohagheghi, P., Dehlen, V., Staron, M., Bouhours, C., Weil, F., Bastarrica, C., Rivas, S., Rossel, P.O., Kuzniarz, L.: Towards a Unified Quality Model for Models. 2nd QiM, RR in Soft. Eng. and Management 2008:01, Gothenburg University

Description and Implementation of a UML Style Guide

Mohammed Hindawi, Lionel Morel, Régis Aubry, and Jean-Louis Sourrouille

Université de Lyon
INSA Lyon, LIESP, Bât. B. Pascal, 69621 Villeurbanne, France
{Mohammed.Hindawi,Lionel.Morel,Régis.Aubry,
Jean-Louis.Sourrouille}@insa-lyon.fr

Abstract. Model quality is still an open issue, and a first step towards quality could be to establish and use style guides. A style guide is a set of rules aiming to help the developer improving models in many directions such as good practices, methodology, consistency, modeling or architectural style, conventions conformance etc. First, this paper attempts to clarify the meaning of notions being used such as rule or modeling domain semantics. Then, several examples illustrate a possible classification of rules, and the verification process is detailed. A style guide is not universal: each project manager should be able to customize his/her set of rules according to specific needs. In addition to rules expressed in OCL, we describe a user interface to facilitate the specification of rules based on quantifiers, along with the translation of these rules into OCL.

1 Introduction

In the emerging context of Model Driven Engineering, software development more and more focuses on models. On the other hand, the software engineering community has known for a long time the advantages of early fault detection. Thus to check models from the beginning of the development cycle appears a promising direction. Generally, application domain semantics is a matter for users, while tools know the semantics of the modeling domain only. Beyond faults, which are all the more difficult to find that models are imprecise and abstract, many model properties are of interest for developers. A style guide is a set of rules aiming to help the developer improving models in many directions such as good practices, methodology, consistency, modeling or architectural style, conventions conformance etc. Some rules are hints while others are warnings, i.e., potential errors. These rules check "good properties", which are kinds of quality criteria. However, the quality of a model is relative to application requirements, which means that deficiencies are ignored as long as they remain in the range defined by the quality objectives of the application. Conversely, a style guide checker notifies all the rule violations: the developer defines his/her own objectives and priorities, often based on error gravity.

Developers could be in charge of rule checking. However, in practice, only automated checks are suitable not to increase developer burden, but also because manual checks are unsure. Consequently, as many rules as possible should be given a formal description, and only rules expressed in natural language will require manual checks. There is no universal style guide. Each development team may have its own needs depending on applications; hence we need an easy way to specify rules. UML provides OCL as a description language, but non-experts find it difficult to use. This implies the need for specific tooling to describe and manage a set of rules, and to control the verification process that should be as flexible and automated as possible.

The rest of the paper (extended version in [17]) is organized as follows: section 2 gives definitions and attempts to clarify the meaning of used notions; section 3 classifies some rule descriptions; section 4 shows the verification process, defines the main tool components, and details the user interface to specify rules including the translation into OCL. Then we discuss related works and conclude.

2 Context and Definitions

This section describes the context of the work and defines notions used in the rest of the paper. Moreover, we aim to clarify what it means to apply rules to a model.

2.1 Syntactic vs. Semantic Correctness

In software engineering, a model is a representation, from a given point of view, of a system expressed in some formalism [4]. The formalism definition includes notions and their semantics. This semantics induces constraints on the model, for instance the semantics of inheritance induces that cycles are not allowed along the inheritance relationship. A model expressed in a formalism is *correct* when it conforms to all the constraints of this formalism. The UML specifies constraints in both OCL and natural language. We call the former *syntactic* constraints and the latter *semantic* constraints [14][6]. Although surprising, this definition is precise and deals with the lack of unambiguous difference between semantic and syntactic constraints (readers may think of semantic constraints as constraints specified in natural language). A model that meets syntactic constraints is *syntactically correct*, and a model that meets semantic constraints is *semantically correct*. Syntactic constraints can be checked automatically while semantic constraints are left to human users, hence it is not possible to check automatically whether a model is correct or not. In everyday cases, the semantic correctness of UML models is unknown. In addition, semantic variation points specified in the UML specification require human choices.

In the following, we consider only syntactically correct models. Additional constraints aim to increase the semantic correctness.

2.2 Interpretations

A system can be modeled in different ways. A model *interpretation* is the meaning of this model in a semantic domain. A correct model has generally several interpretations in a semantic domain (Fig. 1), while an incorrect model has no interpretation. The natural semantic domain of a modeling language such as UML is the *modeling domain*. There is no consensus about a universal semantics of modeling domains; hence, we assume that there are several modeling domains, each one with its own semantics, e.g., active objects do not behave the same according to modeling contexts. The UML does not meet all modeling needs. On the other hand, it allows expressions that the semantics of modeling domains may forbid, e.g., to send a signal to an object that cannot catch it. The modeling domain is not to be mixed up with the application domain. In the modeling domain, a class *Dog* may inherit from a class *Bird*, but in the application domain, this inheritance relationship is surely wrong.

An interpretation may be *licit* in a semantic domain, i.e., it conforms to the semantics of this domain, and illicit in another one. A UML model can be both correct and illicit. For instance, a *TypedElement* without *Type* is correct in UML, but when the element is the receiver of a message, it is illicit in most modeling domains.

Refinement. An abstract model has a large number of interpretations due to the lack of details. Along the development process, models are refined and become more and more complete and precise; hence, the number of interpretations decreases (Fig. 2a). For example, navigability restriction to an undirected association reduces the interpretations. At the end of the refinement process, one interpretation is selected to generate code. This code is a model whose interpretations form an equivalence class from the developer point of view, i.e., all the interpretations are equivalent: the expression *a*+*b*+*c* can be interpreted as either (*a*+*b*)+*c* or *a*+(*b*+*c*).

Checking models. Within a modeling domain, a model is *consistent* when it has at least one licit interpretation. There is no formal definition of the semantics of modeling domains, but many works propose consistency rules to check that models meet some modeling domain constraints. Constraints expressed in a formal language are easy to check. The outstanding issue is how to check models to meet semantic constraints? There is no basic difference between semantic constraints from UML and from modeling domains: both are expressed in natural language, and both aim to reject models, either incorrect or with no licit interpretation. To check these constraints, a first idea is to define *consistency rules* that are stronger than the actual semantic constraints, but that we can express in a formal way. These rules reduce the expressive power of UML (Fig. 2b), i.e., reject potentially correct models and forbid some interpretations, but in return they allow automating checks. A second idea is to define rules that will help the developer to make well-formed models. These rules forbid model expressions leading *generally* to models with illicit or questionable interpretations. In the last resort, human reviews find remaining problems. At code

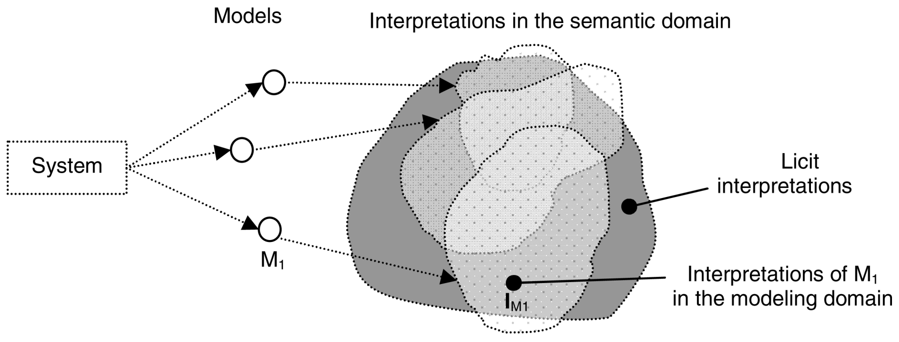

Models
Interpretations in the semantic domain
System
M_1
Licit interpretations
Interpretations of M_1 in the modeling domain
I_{M1}

Fig. 1. Relationships between System, Models and Interpretations

generation, model analysis may reveal errors but it is too late. The checking process fails when an error that was visible in a model is discovered at run time.

2.3 Style Guide

A style guide defines a set of rules that any model must conform to. Style guides reduce the number of acceptable models and force developers to make models owning the wished properties, which results in smaller sets of licit interpretations (Fig. 2b). Unlike language hard constraints, bypassing rules that are simple hints is allowed.

Checking model conformance to a style guide is usually a human issue. Due to the high cost of human reviews, the return on investment seems unsure. To reduce the human burden, this paper details an approach to describe and implement an automatically checkable style guide. We define an architecture on top of existing tools, and a checking process.

2.4 Style Guide and Quality

A style guide defines the boundaries of the set of models owning the wished properties. It aims to help developers designing models with a better quality, for instance using good practices. Of course, the quality of a model that does not meet all the style guide rules is not definitely low; hence, the link between rule violation and quality is to be clarified. Within a multilevel framework for quality assessment such as ISO9126 [8] or[11], rule violation is at the lower level of metrics. Metrics are aggregated to form attributes, which in turn are aggregated to form characteristics. The quality of a model is not subject to conformance to some individual rules, but rather to some statistical knowledge embodied as threshold values for attributes and characteristics. These thresholds come from quality objectives that are set according to specific needs of applications. From the quality point of view, only deviations from these values will lead to corrections, otherwise the model is considered to have the expected quality. While the style guide notifies all rule violations, non-quality is detected only when the combination of a set of metrics reach critical thresholds.

Both style guide and quality assessment detect failures, i.e., non-quality, but nobody knows to what extent a model is good. To ease model comparison, each rule has

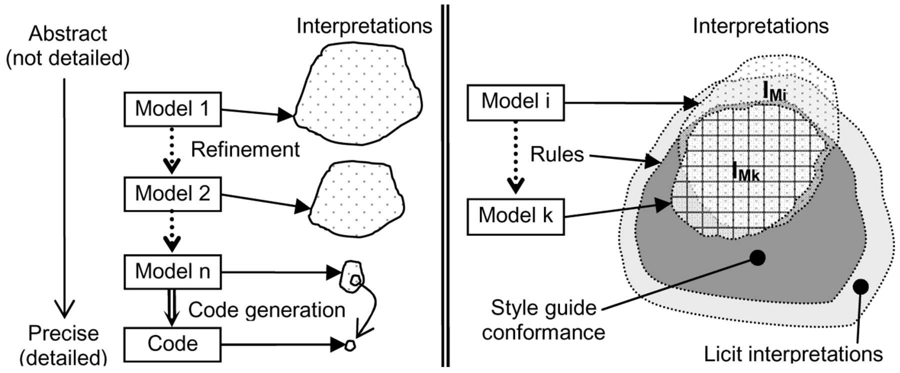


Fig. 2. (a) Model evolution and interpretations (b) Reduction of interpretations

a gravity from warning to serious. Developers know that no serious error should remain in the end while warnings are acceptable.

In spite of the high theoretical maturity level of code quality, it remains an under-exploited way to improve software. The main lessons learned from surveys show that quality should be provided at no cost, with a suitable support, and should not induce delays in the project. Therefore we should pay a great attention to the implementation of the style guide: integration of the verifier within the modeling tool, very simple checking process, flexible user interface, easy rule description, etc.

3 Rules

3.1 Identifying and Classifying Rules

Models are checked along several *dimensions* corresponding to different software engineering areas such as methodology, good practices, or modeling. The semantics of each area induces rules. Since this semantics is generally not described, experts from these areas are in charge of rule identification. In addition to the dimension, which is our main structuring property, each rule owns a set of properties aiming to give further comments, to specify gravity, to link it with model parts or development process stages, to specify and implement it, to describe correction actions, etc. The rules below are classified by dimension, although they might often be attached to several dimensions. As mentioned above, some syntactic rules can be stronger than the actual semantic constraints to allow writing them in OCL. Rules' description is deliberately short and sometimes imprecise due to available space:

- **Methodology** rules come from method description, e.g., "Any communication between actors and subsystem goes through an interface class" (USDP [9]). This rule aims to contain changes to a set of well-identified classes when communication protocols between actors and subsystem are modified.

  Within the development process, methodologies distinguish steps or phases such as requirement elicitation, elaboration, or detailed design. Whatever the methodology, these phases are required to identify moments in the life cycle of artifacts, therefore levels of abstraction. The phase is used to select the set of rules to be applied to each part of a model at a given moment.
- **Common methodology** gathers rules that apply whatever the methodology. They come from skills of experienced developers, e.g.: "A black box sequence diagram only holds actors and a subsystem (definition)" or "A black box sequence diagram only holds communications between actors and a subsystem, not between actors".
- **Consistency** rules detect meaningless expressions in the modeling domain, e.g., "The initial stimulus in a sequence diagram is triggered by an *Actor* or a *Port*, i.e., neither a class instance nor a *Component*". Based on redundancies in the model, some rules detect inconsistencies, e.g., "Within a sequence diagram, actor-to-subsystem interactions should correspond to associations between actors and use cases of this subsystem".
- **Modeling style** rules detect expressions that are *generally* meaningless in the modeling domain. Unlike consistency rules, breaking these rules is tolerated, e.g., "Within any complete class model, a path through navigable associations should

link the root class to any class (not a database schema)". This rule requires marking the *root* class in the model. The rule "Each *ConnectableElement* (from metamodel) in the sequence diagram should be either a port or a class instance" reduces the expressive power forcing components to be connected through ports.

- **Completeness** rules check missing elements, e.g., "When the subsystem *B* is an output actor of the subsystem *A*, then *A* should be an input actor in the description of the subsystem *B*", or "Each association actor-to-use case should be implemented in at least one sequence diagram describing a scenario of this use case".
- **Good practices** rules are often hints, e.g., "Cycles along class associations are to be avoided" which aims to reduce coupling, or "To specify systematically bidirectional navigability for associations is likely unnecessary".
- **Conventions** rules are group agreements about syntactic forms, e.g., "When the class of an attribute is represented on the same diagram, drawing the association is mandatory" to avoid hidden associations.
- **Architecture style** rules aim to aid developers to meet software architecture styles such as low-coupling/high-cohesion or Model-View-Controller, e.g., "A view knows its model but the model does not know its views".
- **Refinement** and **trace** related rules check consistency along the development cycle and enforce links between model elements, e.g., "A sequence diagram should be associated with a use case or a less detailed sequence diagram (traceability)".
- **Specification gap** rules deal with non-standard UML. Modeling tools often allow expressions that do not conform to the UML specifications. To deal with this issue, a set of tool specific rules fulfills the gap between the tool and standard UML specifications. This dimension avoids mixing up style guide rules with UML syntactic constraints that tools do not check.

Since rules check a wide variety of issues, their violation does not have the same gravity. We define three categories: *error*, *warning* and *hint*. When a model has no meaning in the domain of modeling, the violation is an error to be corrected. A violation that might result in a further problem is a warning. When rules such as methodology are hints to improve the modeling process, to correct them after model completion is not always desirable. Although there is a strong link between dimensions and gravity, the gravity is not attached to the dimension.

3.2 Expressing Rules

First, rules are expressed in natural language. Next, they have to be formulated in a formal language, preferably OCL, but non-experts generally find it difficult to use. To allow non-experts to formalize rules, we propose a graphical approach on top of OCL that rely on well-known notions of first-order logic quantification. The rule "Each connected element in a white box sequence diagram should be either a port or an instance of a class" is written:

$\forall x \in \mathrm{WBSeqDiag},\ \mathrm{IsConnectableElem}(x) \Rightarrow \mathrm{IsPort}(x) \vee \exists y,\ \mathrm{IsClass}(y) \wedge x.\mathrm{class}=y$

where *WBSeqDiag* is a collection of model elements, *IsT*(*x*) is a predicate which is true when *x* is an instance of *T*. The general form of this rule is:

$\forall x \in X,\ R_1(x) \Rightarrow R_2(x) \vee \exists y,\ R_3(y)$

Based on quantifiers, our interface provides a limited set of standard forms that ease description but whose expressive power is lower that the OCL one. The main remaining issue is the link between model elements such as class or interaction, and UML metamodel notions. To read and understand the meta-model is hard and reserved to UML experts. In the implementation section, we propose an approach based on rewriting rules that makes it easier to use metamodel notions.

Anyway, to avoid specifying the same rules again and again, a set of standard/ common built-in rules should be provided by tools implementing the style guide. Thus, only specification gap rules and customized rules are to be specified.

4 Verification Process

The verification process lies on the architecture illustrated in Fig. 3. Rules to check depend on the role of the user, the phase in the development process, temporary choices of the developer, etc. *Configurations* express links between rules, model and users and guide the verification process. To check the style guide and to ensure traceability, we need to annotate the model with data such as the current phase or the actor trigger (Fig. 4). Besides, the implementation of the verification process requires marking models to specify which rules should be checked, e.g., a developer knowing that its model is incomplete is not interested in the related errors. To summarize, we need two types of model tags: adornments to complete the model description, and error-processing tags to control the verification process.

4.1 Architecture and Process

The Fig. 3 gives the main processes and data of the verification process:

- **Rules** are managed through an input interface. Rule properties are stored in a description file. The mapping dictionary maps rule concepts to metamodel notions or OCL expressions (detailed further table 1).
- **Configurations** link together a model and sets of rules. These sets are defined either using rule properties, e.g., phase=*Elaboration* and gravity= *error*, or manually for specific needs. For instance, a team member may decide to keep watch on a particular set of rules.

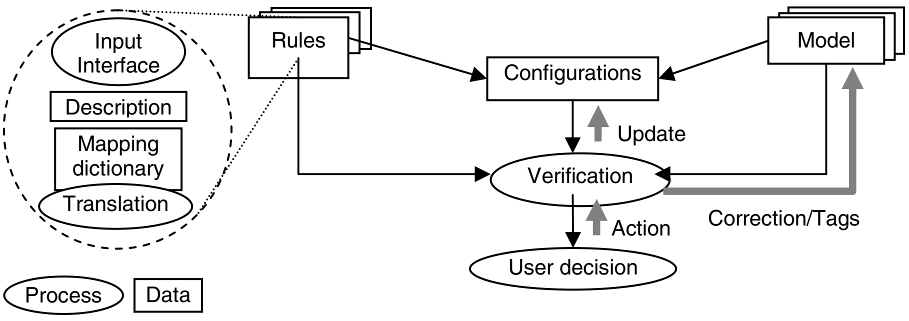


Fig. 3. Verification process

- The **verification** checks rules based on the configurations. When a check fails, data about the rule and related model elements are displayed. According to the rule, several choices are possible: to annotate the model with a tag, to invalidate the rule either in the configuration or in the model, to correct automatically the model. Let us take an example of rule with two diagnoses: "A use case is triggered by only one actor". When no trigger is specified, checking results in a warning for incomplete model. When two triggers are specified (Fig. 4), checking results in an error. The *diagnosis* depends on the number of triggers: 0 is a warning, and greater than 1 an error. This solution avoids several descriptions of the same rule, but each diagnosis holds its own message, gravity, correction, etc.

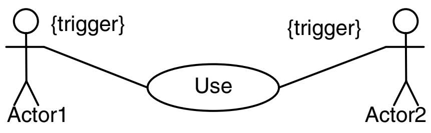
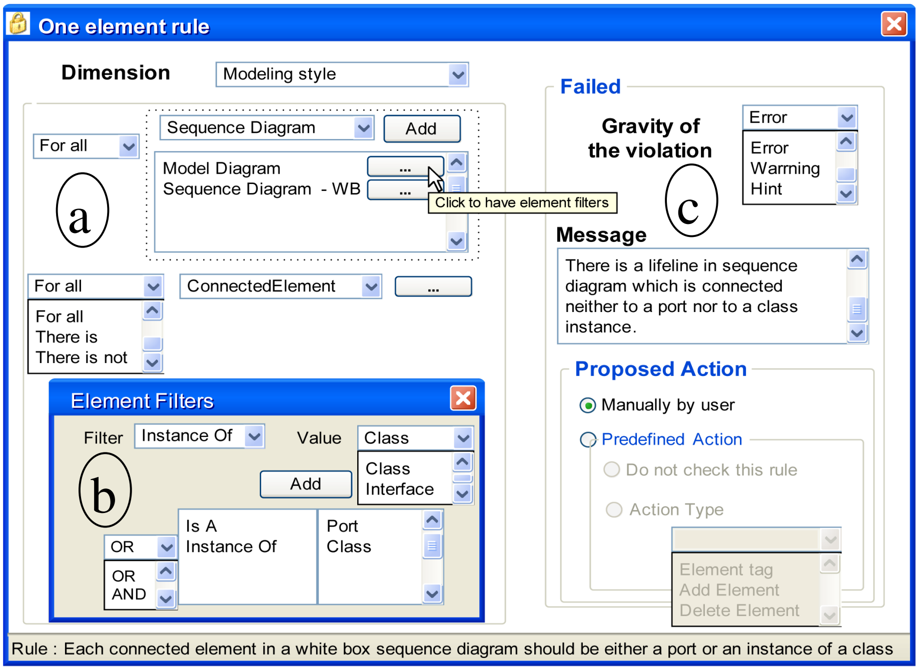


Fig. 4. Rule violation: two triggering actors

4.2 Implementation

To check easily the rules and to aid correction, the style guide is embodied in a IDE (ongoing work on top of Eclipse). The integration into a tool reduces the cost of training and use. We need plug-in extensions to manage configurations, errors, and corrections using model transformations, but the awkward point is rule description.

This section focuses on the design of a user interface aiming to hide the trickiest aspects of OCL. To provide a simplified description of constraints in OCL while keeping the same expressive power is difficult. Our approach is a compromise



One element rule

Dimension: Modeling style

For all — Sequence Diagram — Add

Model Diagram ...
Sequence Diagram - WB ...

Click to have element filters

For all — ConnectedElement — ...

For all / There is / There is not

Element Filters

Filter: Instance Of — Value: Class — Add

Class / Interface

OR — Is A / Instance Of — Port / Class

OR / AND

Failed

Gravity of the violation: Error (Error / Warrning / Hint)

Message: There is a lifeline in sequence diagram which is connected neither to a port nor to a class instance.

Proposed Action
[✓ Manually by user]
Predefined Action
Do not check this rule
Action Type
Element tag / Add Element / Delete Element

Rule : Each connected element in a white box sequence diagram should be either a port or an instance of a class

Fig. 5. Rule description interface

between expressive power and simplicity: simple constraints are specified through the provided interface, while intricate ones are to be written directly in OCL. The main form of the user interface (Fig. 5) allows to expressing a subset of all the possible OCL expressions only. Let us illustrate the description of the rule: "Each connected element in a white box sequence diagram should be either a port or an instance of a class". An equivalent expression using quasi-natural language could be: "For any white box sequence diagram *wb*, for any connected element *e* in *wb*, either *e* is a *Port* or *e* is an instance of a class". This later expression is close to first-order logic and its structure fits well with our generic input form that reads as follow:

For any *Sequence diagram* in *Model diagrams* **such as** *White box* is true

For any connected element *e* , *e* is a *Port* **or** *e* is an instance of a *Class*

The next step is the translation into OCL. Quantifiers are translated into OCL operations such as *forall*, *select, exists*, etc. Notions such as *Sequence diagram* or *Class* are to translate into notions of the UML metamodel.

Translating: For any *Sequence diagram* in *Model diagrams* such as *White box* is true

UML does not supply the notion of diagram: sequence diagrams are *Interaction* owned by packages. From *Package*, the interactions are (Fig. 6):

```
    self.ownedMember->select( i | i.oclIsKindOf(Interaction) )
```

We extract model packages from the metaclass *NamedElement*:

```
NamedElement::allPackages(): Set(Package) ; -- standard operation
    allPackages = NamedElement.allInstances->select( p | p.oclIsKindOf(Package) )
```

Selecting *Interaction*s:

```
NamedElement:: sdFilter() : Set(Interaction) ;
    sdFilter = allPackages()->iterate(p ; result :Set(Interaction)={} |
                    result->union(p.ownedMember->select( i | i.oclIsKindOf(Interaction) ) ) )
```

White Box is not a UML notion, we assume that *GetKindOf*('WhiteBox') returns *true* for a WhiteBox *Interaction*. The set of *White Box* sequence diagrams is:

```
NamedElement:: sdWBFilter() : Set(Interaction) ;
    sdWBFilter = sdFilter()->select( i | i.GetKindOf('WhiteBox') )
```

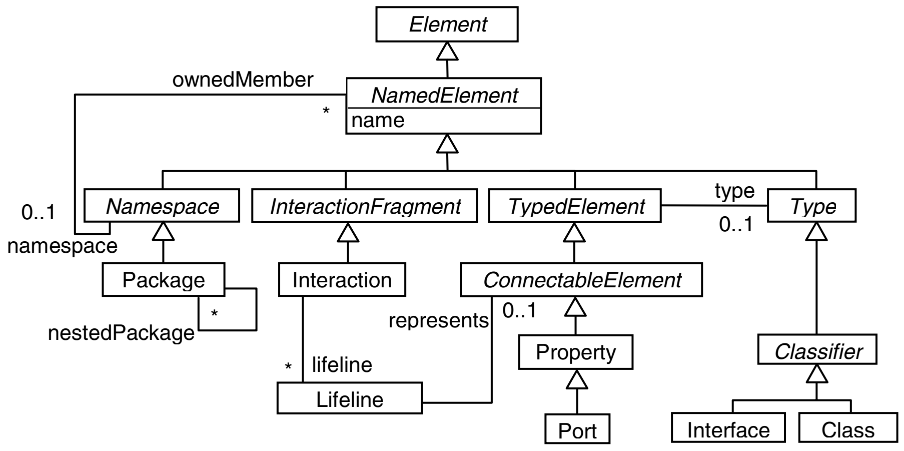


Fig. 6. Root of the required metamodel (from [15])

Table 1. Mapping dictionary

Name	OCL expression
Model diagrams	NamedElement:: Rule() : Boolean ; R1 = allPackages() - - built-in operation, diagrams are owned by packages
Sequence diagram	R2 = R1->iterate(p ; result :Set(Interaction)={} | result->union(p.ownedMember->select(i | i.oclIsKindOf(Interaction))))
White Box	R3 = R2->select(i | i.GetKindOf('WhiteBox'))
Connected element	Rule = R3.lifeline->forAll(f ; x:ConnectableElement= f.represents | R5)
Is A	R5a = x.oclIsKindOf(Port)
Instance Of	R5b = x.type.oclIsKindOf(Class)
or	R5 = R5a **or** R5b

Translating: For any connected element *e*, *e* is a *Port* **or** *e* is an instance of a *Class*

An *Interaction* accesses to its *ConnectableElement* via *represents*. Either the *ConnectableElement* is a *Port* or it is a *Property* whose *type* is a *Class*:

```
    oclIsKindOf(Port)  or type.oclIsKindOf(Class)
```

From *Interaction*, the complete expression is:

```
    lifeline->forAll( f |
              f.represents.oclIsKindOf(Port)  or f.represents.type.oclIsKindOf(Class)   ) )
```

OCL final constraint:

```
NamedElement:: Rule() : Boolean ;
    Rule = sdWBFilter()->forAll( i | i.lifeline->forAll( f |
              f.represents.oclIsKindOf(Port)  or f.represents.type.oclIsKindOf(Class)   ) )
```

To avoid any syntactic error, the interface is aided and users select values in lists, e.g., when *Model Diagrams* is selected (Fig. 5a), the next filter list supplies only allowed subsets. When *Sequence diagram* is selected, the filter allows *WhiteBox*. The translation of the selected expressions into OCL is based on a rewriting principle (Table 1): each element of a list has a value in the dictionary. We plan to build the dictionary using an interface that lists the accessible item names in a context. For instance in the metamodel Fig. 6, from *Interaction* the three only choices are *name*, *lifeline* and *namespace*. This work is close to the definition of a subset of UML [16]. The right area of the interface (Fig. 5c) deals with additional properties and corrective actions when the checked element does not conform to the rule.

5 Related Works

Style guide rules come from various sources. The UML specification [15] is a useful source. UML books such as [1][2] include recommendations or style guides that help making "better" models. Methodology books such as USDP [9] provide tips and rules. Modeling conventions in [12] correspond to several dimensions within our classification. These modeling conventions proved to be useful to reduce model "defects", which confirms that a style guide is an important issue. In addition, papers related to rules or metrics for UML models are interesting sources [13].

A tool may enforce frozen built-in rules, which relieve from the burden of rule description but prevent customization. Using templates, for instance related to a methodology confine the user in a frame, but remaining dimensions are not checked [7]. To summarize, templates enforce a subset of the required rules only, therefore a preferable way will be to include this subset into a more flexible solution.

The verification of style guides described in natural language within books such as [1] must be done manually. Works aiming at automating the verification process should express rules in a formal language. The automated verification on demand is the best solution but proposals are still rare [5][7]. In [7], a checker prototype fully automatically verifies models from rules described using a specific language. Although rule description is different, this work is close to our project. We agree with [5] that find it difficult to write rules in OCL. Instead of defining a new language as in [7], we provide a user interface to aid specifying rules that are next translated into OCL. This way we keep a standard language while aiding rule description. In this direction, some works aim to facilitate OCL writing: VisualOCL [3][10] visualizes OCL expressions in an alternative notation. It provides additional information, which increases the usability of OCL. However, to use such tool implies experience in OCL. We try to overcome this issue by proposing an interface easy to use, at a high abstraction level, but rather far from OCL, which implies an additional and tricky translation process.

6 Conclusion

This project is under development[1] and some issues are pending. The advance of our solution lies in the integration of several technical artifacts to form a complete methodology and tooling. This integration associated with automated checking and style guide customization is a necessary condition for actual use in companies. Some particularly relevant elements in our approach include:

- Selective checking of model parts using tags, which avoid re-checking of rules and messages related to incomplete model parts, therefore lighten the user burden;
- Selective checking according to the current phase in the methodology;
- Customization of the set of active rules in a configuration file according to developer role and experience, application domain, expected "quality", etc.
- Aid for correcting models: when a rule is violated, the developer may choose a predefined action including model change by applying patterns;
- Aid for defining rules: the graphical interface helps project managers in the definition of rules for their own style guide.

This work is part of a grant aiming to assess model quality. The companies involved in the project will help us to tune quality assessment from metrics. Model quality assessment is relative to application quality requirements and developers do not always know the important quality criteria. A style guide brings the educational

[1] Partly financed by the grant PACTE QUALITE with the Rhône-Alpes regional government.

aspect needed to help increasing models' "good properties": it detects all rules violations but also provides hints, warns to avoid potential errors, and may include company know-how. Finally, a style guide is a quite necessary complement to put into practice quality assessment.

References

1. Ambler, S.W.: The Elements of UML 2.0 Style. Cambridge University Press, Cambridge (2005)
2. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, Reading (1998)
3. Bottoni, P., Koch, M., Parisi-Presicce, F., Taentzer, G.: A visualization of OCL using collaborations. In: Gogolla, M., Kobryn, C. (eds.) UML 2001. LNCS, vol. 2185, pp. 257–271. Springer, Heidelberg (2001)
4. Caplat, G., Sourrouille, J.L.: MDA: Model Mapping using Formalism Extension. IEEE Software 22(2), 44–51 (2005)
5. Farkas, T., Hein, C., Ritter, T.: Automatic Evaluation of Modeling Rules and Design Guidelines. In: Proc. of the Workshop From code centric to Model centric Soft. Eng., `http://www.esi.es/modelware/c2m/papers.php`
6. Harel, D., Rumpe, B.: Modeling Languages: Syntax, Semantics and All That Stuff, TR MCS00-16, The Weizmann Institute of Science (2000)
7. Hnatkowska, B.: Verification of Good Design Style of UML Models. In: Proc. Int. Conf. Information System Implementation and Modeling (2007)
8. ISO, International Organization for Standardization, ISO 9126-1:2001, Software engineering – Product quality, Part 1: Quality model (2001)
9. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Reading (1999)
10. Kiesner, C., Taentzer, G., Winkelmann, J.: Visual OCL: A Visual Notation of the Object Constraint Language. Technical Report 2002/23, Tech. Univ. of Berlin (2002)
11. Kuzniarz, L., Pareto, L., Sourrouille, J.-L., Staron, M.: Third intenational workshop on quality in modeling. In: Giese, H. (ed.) MoDELS 2007 Workshops. LNCS, vol. 5002, pp. 271–274. Springer, Heidelberg (2008)
12. Lange, C.F.J., DuBois, B., Chaudron, M.R.V., Demeyer, S.: Experimentally investigating the effectiveness and effort of modeling conventions for the UML. CS-Report 06-14, Tech. Univ. Eindhoven (2006)
13. Malgouyres, H., Motet, G.: A UML model consistency verification approach based on meta-modelling formalization. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 1804–1809. Springer, Heidelberg (2007)
14. Sourrouille, J.-L., Caplat, G.: A Pragmatic View about Consistency Checking of UML Model, Work. Consistency Problems in UML-Based Software Dev., pp. 43–50 (2003)
15. UML, OMG Unified Modeling Language, Version 2.1.2 (2007)
16. Sourrouille, J.L., Hindawi, M., Morel, L., Aubry, R.: Specifying consistent subsets of UML. In: Proc. Educators Symposium, Models 2008, pp. 26–38 (2008) ISBN 83-916444-8-0
17. Hindawi, M., Morel, L., Aubry, R., Sourrouille, J.L.: Description and Implementation of a Style Guide for UML. In: Proc. 3rd Workshop on Quality in Modeling, IT Univ Göteborg RR 2008:02, pp. 31–45 (2008) ISSN 1654-4870

Empirical Validation of Measures for UML Class Diagrams: A Meta-Analysis Study

M. Esperanza Manso[1], José A. Cruz-Lemus[2], Marcela Genero[2], and Mario Piattini[2]

[1] GIRO Research Group, Department of Computer Science, University of Valladolid, Campus Miguel Delibes, E.T.I.C., 47011, Valladolid, Spain
manso@infor.uva.es
[2] ALARCOS Research Group, Department of Technologies and Information Systems, University of Castilla-La Mancha, Paseo de la Universidad, 4, 13071 Ciudad Real, Spain
{JoseAntonio.Cruz,Marcela.Genero,Mario.Piattini}@uclm.es

Abstract. The main goal of this paper is to show the findings obtained through a meta-analysis study carried out with the data obtained from a family of five controlled experiments performed in academic environments. This family of experiments was carried out to validate empirically two hypotheses applied to UML class diagrams, which investigate 1) The dependence between the structural complexity and size of UML class diagrams on one hand and their cognitive complexity on the other, as well as 2) The dependence between the cognitive complexity of UML class diagrams and their comprehensibility and modifiability. We carried out a meta-analysis, as it allows us to integrate the individual findings obtained from the execution of a family of experiments carried out to test the aforementioned hypotheses. The meta-analysis reveals that the measures related to associations and generalizations have a strong correlation with the cognitive complexity, and that the cognitive complexity has a greater correlation to comprehensibility than to modifiability. These results have implications from the points of view of both modeling and teaching, revealing which UML constructs are most influential when modelers have to comprehend and modify UML class diagrams. In addition, the measures related to associations and generalizations could be used to build prediction models.

Keywords: meta-analysis, experiments, UML class diagrams, comprehensibility, modifiability, structural complexity, size.

1 Introduction

The Model-Driven Development paradigm (MDD) [1] is an emerging approach for software development which is of ever-increasing interest to both the research community and software practitioners. MDD considers models as end-products rather than simply as means to produce software. In this context the quality focus has shifted from code to models, given that the quality of the models obtained through transformations is of great importance. This is because it will ultimately determine the quality of the software systems produced. Since, in the context of MDD, maintenance must be done on models, we are concerned about sub-characteristics of maintainability, such as the comprehensibility and modifiability of UML class diagrams. Class diagrams constitute the backbone of a system design and they must be comprehensible and flexible enough to allow the modifications that reflect changes in the things they model to be incorporated easily. We have based our work on the model shown in Figure 1 [2, 3]. This model constitutes a theoretical basis for the development of quantitative models relating to internal and external quality attributes and has been used as the basis for a great amount of empirical research into the area of structural properties of software artefacts [4-6]. In the study reported here, we have assumed a similar representation for UML class diagrams. We hypothesize that the structural properties (such as structural complexity and size) of a UML class diagram have an effect on its cognitive complexity. Cognitive complexity can be defined as the mental burden placed by the artefact on the people who have to deal with it (e.g. modellers, maintainers). High cognitive complexity will result in the production of an artefact that has reduced comprehensibility and modifiability, which will consequently affect its maintainability.

The main motivation behind the research we have been carrying out is to validate this model, formulating two main hypotheses based on each of the arrows in Figure 1:

Size and structural complexity of UML class diagrams affect cognitive complexity

Cognitive complexity affects the comprehensibility and modifiability of UML class diagrams.

To measure the content of each box of Figure 1 we have defined some measures, which will be introduced in Section 3. In order to test such hypotheses, we carried out 5 experiments, which constitute a family of experiments [7, 8].

The data analysis carried out in each individual experiment did not allow us to obtain conclusive results. This led us to carry out a meta-analysis study. Meta-analysis has been recognised as an appropriate way to aggregate or integrate the findings of empirical studies in order to build a solid body of knowledge on a topic based on empirical evidence [9-11]. Moreover, the need for meta-analysis is gaining relevance in empirical research, as is demonstrated by the fact that it is a recurrent topic in various forums related to Empirical Software Engineering. Meta-analysis is a tool for extracting these global conclusions from families of experiments, as it allows us to estimate the global effect size of the whole family, as well as to measure the accuracy of this measure and to evaluate the significance of effect size with respect to the hypotheses under study.

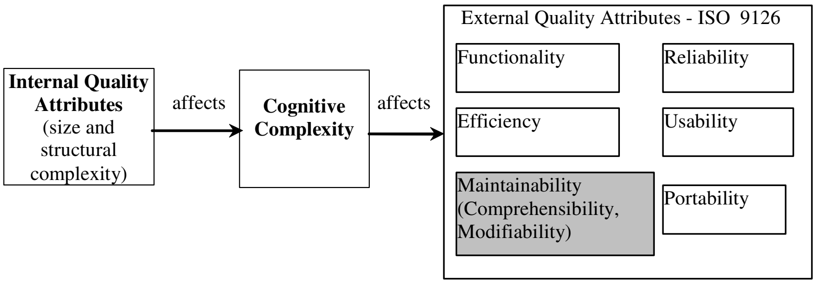

Internal Quality Attributes (size and structural complexity)
affects
Cognitive Complexity
affects
External Quality Attributes - ISO 9126
Functionality
Reliability
Efficiency
Usability
Maintainability (Comprehensibility, Modifiability)
Portability

Fig. 1. Relationship between structural properties, cognitive complexity, and external quality attributes, based on [1, 3]

The main goal of the current paper is to present a meta-analysis study that would serve to integrate the results obtained from previous experimentation. In this way, meta-analysis contributes to the obtaining of a solid body of knowledge concerning the usefulness of the measures for UML Class diagrams.

The remainder of the paper is organised as follows: Section 2 describes the family of experiments. The Meta-analysis study is presented in Section 3. Finally, the last section presents some concluding remarks and outlines our future work.

2 The Family of Experiments

Isolated studies (or experiments) hardly ever provide enough information to answer the questions posed in a research study [10, 12, 13]. Thus, it is important for experiments to be part of families of studies [12]. Common families of studies allow researchers to answer questions that are beyond the scope of individual experiments, and to generalize findings across studies. In this work we will comment on five experiments, whose main contextual characteristics are summarized in Table 1.

Table 1. Characteristics of the experiments

Study	#Subjects	University	Date	Year
E1	72	University of Seville (Spain)	March 2003	4th
R1	28		March 2003	
E2	38	Univ. of Castilla-La Mancha (Spain)	April 2003	3rd
R21	23	University of Sannio (Italy)	June 2003	4th
R22	71	University of Valladolid (Spain)	Sept. 2005	3rd

To perform the experiments, we followed the guidelines provided in [14, 15].

2.1 Planning of Experiments

In this sub-section we will define the common framework of all the studies:

1. **Preparation**. The family has the goal of testing both the hypotheses presented in the introduction.
   - To analyze the structural complexity of UML class diagrams with respect to their relationship with cognitive complexity from the viewpoint of software modelers or designers in an academic context.
   - To analyze the cognitive complexity of UML class diagrams with respect to their relationship with comprehensibility and modifiability from the viewpoint of software modelers or designers in an academic context.
2. **Context definition.** In these studies, we have used students as experimental subjects. The tasks to be performed did not require high levels of industrial experience, so we believed that these subjects might be considered appropriate, as is pointed out in several works [12, 16]. In addition, working with students implies a set of advantages, such as the facts that the students' prior knowledge is fairly homogeneous, a large number of subjects is readily available, and there is the possibility of testing experimental design and initial hypotheses [17]. A further advantage of using novices as subjects in experiments on understandability is that the

cognitive complexity of the object under study is not hidden by the subjects' experience.

3. **Material.** The experimental materials consisted of a set of UML class diagrams suitable for the family goals. The selected UML class diagrams covered a wide range of the metrics values, considering three types of diagrams: Difficult to maintain (D), Easy to Maintain (E) and Moderately difficult to maintain (M). Some were specifically designed for the experiments and others were obtained from real applications. Each diagram had some documentation attached, containing, among other things, four comprehension and four modification tasks.

2.2 How the Individual Experiments were Conducted

We shall now explain the experimental plan of the different members of the family of experiments. The variables considered for measuring the structural complexity and size were the set of 11 measures presented in Table 7 in Appendix A. The *CompSub* measure is the subjective perception given by the subjects with regard to the complexity of the diagrams they have to work with during the experimental task. We consider *CompSub* to be a measure of cognitive complexity. The allowable values of this variable are: Very simple, Moderately simple, Average, Moderately complex and Very complex. To measure the Comprehensibility and Modifiability of UML class diagrams, we considered the time (in seconds) taken by each subject to complete the comprehensibility and modifiability tasks. We called these measures the Comprehensibility and Modifiability time.

We used a counter-balanced between-subjects design, i.e., each subject works with only one diagram. The diagrams were randomly assigned and each diagram is considered by the same number of subjects.

We formulated the following hypotheses, which are derived from the family's goals:

- $H_{0,1}$: The structural complexity and size of UML class diagrams are not correlated with the cognitive complexity. $H_{1,1}$: $\neg H_{0,1}$
- $H_{0,2}$: The cognitive complexity of UML class diagrams is not correlated with their comprehensibility and modifiability. $H_{1,2}$: $\neg H_{0,2}$

All the experiments were supervised and time-limited. More details can be found in [7, 8]. Finally, we used SPSS [18] to perform all the statistical analyses and the tool Comprehensive Meta Analysis [19] was employed to perform the meta-analysis.

2.3 Experiment 1 (E1) and Replication (R1)

On testing the hypotheses we obtained the following findings:

- The correlation between the *CompSub* variable and the 11 metrics was significant at a 0.05 level for E1. We also obtained a significant correlation for R1 in all cases, with the exception of the NM, NGen and MaxDIT metrics.
- The subjective complexity seems to be positively correlated to the effort needed to comprehend UML class diagrams, but the results are significant only for E1 (see Table 2). At the same time, there is no correlation with the effort needed to modify the diagrams. A possible explanation for this could be that the subjects base their perception on the difficulty of the first tasks that they perform, which in this case are the comprehension ones.

Table 2. Results related to goal 2 for E1 & R1

Variables correlated	**E1 (n=62)**		**R1(n= 22)**	
	$\rho_{spearman}$	**p-value**	$\rho_{spearman}$	**p-value**
CompSub vs Comprehensibility	0.266	**0.037**	0.348	0.111
CompSub vs Modifiability	0.132	0.306	0.270	0.217

2.4 Experiment 2 (E2) and its Replications (R21 and R22)

In these studies, goals and variables are the same as in the previous ones, but the diagrams used were different, and context and design have also been improved. More detailed information about them can be found in [8].

Apart from the family's variables, some other variables have been added, in order to validate the results:

- *CompCorrectness* = # correct comprehension tasks / # total tasks performed
- *CompCompleteness* = # correct comprehension tasks / # total tasks to perform
- *ModifCorrectness* = # correct modification tasks / # total tasks performed
- *ModifCompleteness* = # correct modification tasks / # total tasks to perform

Again, we use a between-subjects design, but in this case it has been improved by blocking the subjects' experience. A pre-test was performed, the results of which led to the subjects' being divided into two groups. Each diagram was then assigned to the same number of subjects from each group. More details about this process can be found in [8].

The Comprehensibility and Modifiability measures were only included when the tasks performed had a minimum quality level, and it was for this reason that we used the newly introduced variables, presented previously. The subjects who attained under 75% in correctness and completeness were excluded from the study. In fact their exclusion improved the behaviour of the dependent variables, i.e, symmetry and outliers.

On testing the hypotheses we obtained the following findings:

Table 3. Goal 1 results for E2, R21 & R22

Study	Significantly correlated metrics
E2	NC, NAssoc, NGen, NGenH, MaxDIT (5 out of 11)
R21	All except for NM, NGenH and MaxAgg (8 out of 11)
R22	All except for NM (10 out of 11)

Table 4. Results related to goal 2 for E2, R21 & R22

Variables correlated	E2			R21			R22		
	$\rho_{spearman}$	p-value	N	$\rho_{spearman}$	p-value	N	$\rho_{spearman}$	p-value	N
CompSub vs Comprehensibility	0.343	**0.049**	33	0.410	**0.065**	21	0.353	**0.003**	70
CompSub vs Modifiability	0.337	0.099	25	0.156	0.500		0.165	0.173	

- We have favourable results which admit a correlation between the structural and the cognitive complexities of UML class diagrams. Most of the metrics are significantly correlated with the subjective complexity in the different studies; especially those related to inheritance hierarchies (see Table 3).
- The results are also in favour of the hypothesis that relates cognitive complexity to the comprehensibility of UML class diagrams (see Table 4).

2.5 Threats to the Validity of the Family of Experiments

The main threats to the validity of the family are the following:

- **Conclusions validity.** The number of subjects in R1, E2 and R21 is quite low, and subjects were selected by convenience. Our conclusions must therefore be applied to the population represented by these subjects.
- **Internal validity.** We have found correlation between the variables, which implies the possibility of the existence of that causality, but not the causality itself. Moreover, R21 materials were written in English, which is not the mother language of the subjects (Italians). This fact may have increased the times taken to perform the tasks, especially those of modification.
- **External validity.** It would be advisable to perform some replications with data extracted from real projects, in an effort to generalise the results obtained.

3 Meta-analysis Study

There are several statistical methods that allow us to accumulate and interpret a set of results obtained through different inter-related experiments, since they check similar hypotheses [20-24]. There are three main ways in which to perform this process: meta-analysis, significance level combination and vote counting.

According with the characteristic of our data, in the present study we used meta-analysis, which is a set of statistical techniques that allow us to combine the different effect size measures (or treatment effect) of the individual experiments. There are several metrics to obtain this value, e.g. the means difference and the correlation coefficients, among others [21]. The objective is to obtain a global effect, the treatment effect of all experiments. As effect size measures may come from different environments and may even not be homogeneous, it is necessary to obtain a standardized measure of each one. For example, the dependence between two variables could be measured by different coefficients or scales The global effect size is obtained as a weighted average of standardized measures, in which the most commonly used weights are the sample size or the standard deviation. Together with the estimation of the global effect size, we can provide an estimated confidence interval and a p-value which allows us to decide on the meta-analysis hypotheses. We can find several applications of this technique in Empirical Software Engineering [25, 26].

We have a family of experiments whose main goals are:

1. To study the influence of metrics on the cognitive complexity of UML class diagrams.
2. To study the influence of cognitive complexity on the comprehensibility and modifiability of UML class diagrams.

The use of meta-analysis will allow us to extract global conclusions, despite the fact that some of the experimental conditions are not the same. As we have mentioned previously, we will need to standardize the effect sizes. In this meta-analysis we used correlation coefficients (r_i) that, once transformed (Fisher transformation), provide the effect sizes that have a Normal distribution (z_i), what makes them easier to use. The global effect size is obtained using the Hedges' g metric [21, 27], that is a weighted mean which has the proportional weights to the experiment size (equation 1).

$$\overline{Z} = \frac{\sum_i w_i z_i}{\sum_i w_i} \qquad W_i = 1/(n_i\text{-}3) \tag{1}$$

The higher the value of Hedges' g is, the higher the corresponding correlation coefficient is too. For studies in Software Engineering, we can classify effect sizes into small, medium and large [27]. We rely on the use of the five empirical studies, previously presented in this work, which means that the conclusions about our goals will be extracted from five different results.

3.1 Meta-analysis Results

Firstly, a meta-analysis for each metric-*CompSub* pair will be carried out, taking into account the fact that the hypothesis test is one-tailed, i.e., we consider as null-hypothesis that the correlation is now above zero. In Table 5 we present the global estimation of the correlation coefficient, a confidence interval at 95%, the p-value and the value for Hedges' g, including a classification of the effect size as large (L), medium (M) or small (S).

The results observed are in favour of the existence of a positive correlation between cognitive complexity and the 11 metrics that measure the structural complexity and size of UML class diagrams. In fact, most of the effect sizes are medium or large, with the exception of NM, which is small. The size metrics that have most influence upon the cognitive complexity are NC and NA, while the complexity metrics that have most influence upon cognitive complexity are related to aggregations (NAgg) and generalizations (NGen and MaxDIT). We can conclude that those diagrams with many classes and attributes will have an increased cognitive complexity. Moreover, class diagram models using many inheritance and aggregation mechanisms will also have an increased cognitive complexity.

With regard to the hypotheses derived from goal 2, Table 6 shows that we can admit the existence of correlation between the cognitive complexity and the two measures, Comprehensibility and Modifiability, which measure quality attributes of UML class diagrams.

The effect sizes are medium in both cases, but the correlation estimation of Comprehensibility is larger than the correlation of Modifiability. So we can conclude that, the more cognitive complexity a diagram contains, the more difficult it will be to comprehend and modify.

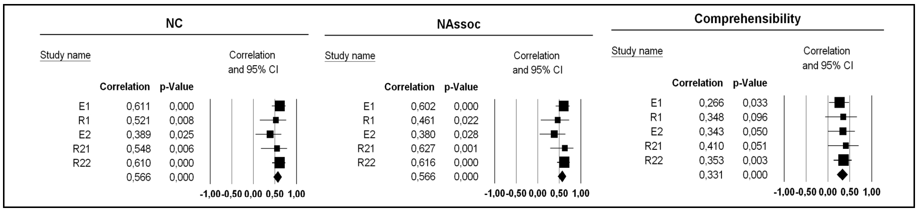
As an example, Figure 2 presents in diagram form the meta-analysis of the relationship of a couple of metrics and the *CompSub* measure, and the relationship between their comprehensibility and cognitive complexity.

Table 5. Meta-analysis of metrics-*CompSub*

H_0: ρ≤0	**Correlation (ρ) Global effect size**	**Lower limit**	**Upper limit**	**p-value**	**Hedges'g**
NC	0.566	0.464	0.653	0.0000	1.322(L)
NA	0.541	0.435	0.632	0.000	1.219(L)
NM	0.177	0.040	0.307	0.012	0.339(S)
NAssoc	0.566	0.465	0.653	0.000	1.318(L)
NAgg	0.481	0.368	0.581	0.000	1.051(M)
NDep	0.484	0.371	0.584	0.000	1.060(M)
NGen	0.484	0.371	0.584	0.000	1.018 (L)
NGenH	0.422	0.302	0.529	0.000	0.903 (M)
NAggH	0.393	0.270	0.504	0.000	0.814 (M)
MaxDIT	0.492	0.379	0.590	0.000	1.080 (L)
MaxHAgg	0.360	0.233	0.474	0.000	0.734 (M)

Table 6. Meta-analysis of *CompSub*-Comprehensibility and Modifiability time

H_0: ρ≤0	**Correlation (ρ) gobal effect size**	**Lower limit**	**Upper limit**	**p-value**	**Hedges'g**
Comprehensibility Time	0.330	0.200	0.449	0.000	0.684 (M)
Modifiability Time	0.186	0.044	0.320	0.011	0.368(M)



NC

Study name	Correlation	p-Value
E1	0,611	0,000
R1	0,521	0,008
E2	0,389	0,025
R21	0,548	0,006
R22	0,610	0,000
	0,566	0,000

NAssoc

Study name	Correlation	p-Value
E1	0,602	0,000
R1	0,461	0,022
E2	0,380	0,028
R21	0,627	0,001
R22	0,616	0,000
	0,566	0,000

Comprehensibility

Study name	Correlation	p-Value
E1	0,266	0,033
R1	0,348	0,096
E2	0,343	0,050
R21	0,410	0,051
R22	0,353	0,003
	0,331	0,000

Fig. 2. Meta-analysis for NC-*CompSub*, NAssoc-*CompSub* and *CompSub-Comprehensibility*

4 Conclusions

The main goal of this work has been that of validating a theoretical model which relates the structural complexity and size of UML class diagrams and cognitive complexity to two of their external quality attributes: comprehensibility and modifiability (Figure 1). For that purpose, we carried out a meta-analysis study with the data obtained from a family of five experiments. The meta-analysis results are in favour of the model under inspection with regard to the two goals being pursued:

- Goal 1: structural complexity is correlated with cognitive complexity, especially with that related to associations and generalizations. An increase in the number of classes and attributes within classes also increases the cognitive complexity of UML class diagrams.

- Goal 2: cognitive complexity influences both the comprehensibility time and modifiability time of UML class diagrams, but this is especially true in the former case.

These results are relevant, as they point to a means of controlling the level of certain quality attributes of UML class diagrams from the modelling phase. The findings also have implications, both practically and in terms of teaching, providing information about which UML constructs may have more implications in the effort to understand and maintain UML class diagrams. When alternative designs of UML class diagrams exist, it could be advisable to select the one which minimizes these constructs.

Moreover, the measures related to associations and generalizations could be used to build prediction models, to evaluate how the time taken to understand or modify an UML class diagram increases; we have done this prediction modeling in [8]. In future work we plan to refine the prediction models obtained, using the data obtained in the whole family of experiments.

Further experimentation is needed to confirm the findings of the current study, improving different issues: 1) Increasing the class diagram sample, 2) Working with practitioners, 3) Improving the modifying tasks and 4) Investigating other metrics to do with cognitive complexity.

Also pending is the carrying out of a similar study with the measures we have defined for UML statechart diagrams [28] and OCL expressions [29].

Acknowledgements

This research is part of the ESFINGE project (TIN2006-15175-C05-05) financed by the “Ministerio de Educación y Ciencia (Spain)”, the IDONEO project (PAC08-0160-6141) financed by “Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha”.

References

1. Atkinson, C., Kühne, T.: Model Driven Development: a Metamodeling Foundation. IEEE Transactions on Software Engineering 20, 36–41 (2003)
2. Briand, L., Morasca, S., Basili, V.: Defining and Validating Measures for Object-Based High-Level Design. IEEE Transactions on Software Engineering 25, 722–743 (1999)
3. ISO-IEC, ISO/IEC 9126. Information Technology - Software Product Quality (2001)
4. El-Emam, K., Melo, W.: The Prediction of Faulty Classes Using Object-Oriented Design Metrics, National Research Council of Canada (1999)
5. El-Emam, K., Benlarbi, S., Goel, N., Rai, S.: The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics. IEEE Transactions on Software Engineering 27(7), 630–650 (2001)
6. Poels, G., Dedene, G.: Measures for assessing dynamic complexity aspects of object-oriented conceptual schemes. In: Laender, A.H.F., Liddle, S.W., Storey, V.C. (eds.) ER 2000. LNCS, vol. 1920, pp. 499–512. Springer, Heidelberg (2000)
7. Genero, M., Manso, M.E., Piattini, M.: Early Indicators of UML Class Diagrams Understandability and Modifiability. In: ACM-IEEE International Symposium on Empirical Software Engineering (2004)
8. Genero, M., Manso, M.E., Visaggio, A., Canfora, G., Piattini, M.: Building Measure-Based Prediction Models for UML Class Diagram Maintainability. Empirical Software Engineering 12, 517–549 (2007)
9. Lipsey, M., Wison, D.: Practical Meta-Analysis. Sage, Thousand Oaks (2001)

10. Miller, J.: Applying Meta-Analytical Procedures to Software Engineering Experiments. Journal of Systems and Software 54, 29–39 (2000)
11. Pickard, L.M.: Combining Empirical Results in Software Engineering, University of Keele (2004)
12. Basili, V., Shull, F., Lanubile, F.: Building Knowledge through Families of Experiments. IEEE Transactions on Software Engineering 25, 456–473 (1999)
13. Shull, F., Carver, J., Travassos, G., Maldodano, J., Conradi, R., Basili, V.: Replicated Studies: Building a Body of Knowledge about Software Reading Techniques. Lecture Notes on Empirical Software Engineering, pp. 39–84. World Scientific, Singapore (2003)
14. Wohlin, C., Runeson, P., Hast, M., Ohlsson, M.C., Regnell, B., Wesslen, A.: Experimentation in Software Engineering: an Introduction. Kluwer Academic Publisher, Dordrecht (2000)
15. Juristo, N., Moreno, A.: Basics of Software Engineering Experimentation. Kluwer Academic Publishers, Dordrecht (2001)
16. Höst, M., Regnell, B., Wohlin, C.: Using Students as Subjects - a Comparative Study of Students & Professionals in Lead-Time Impact Assessment. In: 4th Conference on Empirical Assessment & Evaluation in Software Engineering (EASE 2000). Keele, UK (2000)
17. Sjoberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V., Karahasanovic, A., Liborg, N.K., Rekdal, A.C.: A Survey of Controlled Experiments in Software Engineering. IEEE Transactions on Software Engineering 31(9), 733–753 (2005)
18. SPSS, SPSS 12.0, Syntax Reference Guide. SPSS Inc.: Chicago, USA (2003)
19. Biostat, Comprehensive Meta-Analysis v2 (2006)
20. Glass, G.V., McGaw, B., Smith, M.L.: Meta-Analysis in Social Research. Sage Publications, Thousand Oaks (1981)
21. Hedges, L.V., Olkin, I.: Statistical Methods for Meta-Analysis. Academia Press (1985)
22. Rosenthal, R.: Meta-Analytic Procedures for Social Research. Sage Publications, Thousand Oaks (1986)
23. Sutton, J.A., Abrams, R.K., Jones, R.D., Sheldon, A.T., Song, F.: Methods for Meta-Analysis in Medical Research. John Wiley & Sons, Chichester (2001)
24. Wolf, F.M.: Meta-Analysis: Quantitative Methods for Research Synthesis. Sage Publications, Thousand Oaks (1986)
25. Dybå, T., Arisholm, E., Sjøberg, D.I.K., Hannay, J.E., Shull, F.: Are Two Heads Better than One? On the Effectiveness of Pair Programming. IEEE Software 24(6), 10–13 (2007)
26. Miller, J., McDonald, F.: Statistical Analysis of Two Experimental Studies, University of Strathclyde (1998)
27. Kampenes, V., Dybå, T., Hannay, J.E., Sjoberg, D.I.K.: A Systematic Review of Effect Size in Software Engineering Experiments. Information and Software Technology 49(11-12), 1073–1086 (2007)
28. Cruz-Lemus, J.A., Genero, M., Piattini, M.: Metrics for UML Statechart Diagrams. In: Genero, M., Piattini, M., Calero, C. (eds.) Metrics for Software Conceptual Models. Imperial College Press, UK (2005)
29. Reynoso, L., Genero, M., Piattini, M.: Measuring OCL Expressions: An approach based on Cognitive Techniques. In: Genero, P.A.C. (ed.) Metrics for Software Conceptual Models. Imperial College Press, UK (2005)
30. Genero, M., Piattini, M., Calero, C.: Early Measures for UML Class Diagrams. L'Object 6(4), 495–515 (2000)
31. Genero, M., Poels, G., Manso, M.E., Piattini, M.: Defining and Validating Metrics for UML Class Diagrams. In: Genero, M., Piattini, M., Calero, C. (eds.) Metrics for Software Conceptual Models. Imperial College Press, UK (2005)

32. Chidamber, S., Kemerer, C.: A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering 20, 476–493 (1994)

Appendix A

After studying the UML metamodel, and having reviewed the literature concerning existing measures, we proposed a set of eight measures for the structural complexity of UML class diagrams [30, 31]. The proposed measures are related to the usage of UML relationships, such as associations, dependencies, aggregations and generalizations. In the study reported in this work, we have also considered traditional OO measures, such as size measures (see Table 7).

Table 7. Measures for UML class diagrams

	Measure Name	Measure definition
Size measures	Number of Classes (NC)	The total number of classes in a class diagram.
	Number of Attributes (NA)	The number of attributes defined across all classes in a class diagram (not including inherited attributes or attributes defined within methods). This includes attributes defined at class and instance level.
	Number of Methods (NM)	The total number of methods defined across all classes in a class diagram, not including inherited methods (as this would lead to double counting). This includes methods defined at class and instance level.
Structural complexity measures	Number of Associations (NAssoc)	The total number of association relationships in a class diagram.
	Number of Aggregations (NAgg)	The total number of aggregation relationships (each "whole-part" pair in an aggregation relationship).
	Number of Dependencies (NDep)	The total number of dependency relationships.
	Number of Generalizations (NGen)	The total number of generalization relationships (each "parent-child" pair in a generalization relationship).
	Number of Generalization Hierarchies (NGenH)	The total number of generalization hierarchies, i.e. it counts the total number of structures with generalization relationships.
	Number of Aggregation Hierarchies (NAggH)	The total number of aggregation hierarchies, i.e. it counts the total numbers of "whole-part" structures within a class diagram.
	Maximum DIT (MaxDIT).	The maximum DIT value obtained for each class of the class diagram. The DIT value for a class within a generalization hierarchy is the longest path from the class to the root of the hierarchy [32].
	Maximum HAgg (MaxHAgg)	The maximum HAgg value obtained for each class of the class diagram. The HAgg value for a class within an aggregation hierarchy is the longest path from the class to the leaves.

First Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2008)

Fernando Silva Parreiras[1], Jeff Z. Pan[2], Uwe Assmann[3], and Jakob Henriksson[3]

[1] ISWeb — Information Systems and Semantic Web,
Institute for Computer Science, University of Koblenz-Landau
Universitaetsstrasse 1, Koblenz 56070, Germany
parreiras@uni-koblenz.de

[2] Department of Computing Science, The University of Aberdeen
Aberdeen AB24 3UE
jpan@csd.abdn.ac.uk

[3] Institute for Software- and Multimedia-Technology, TU Dresden
D-01062 Dresden, Germany
{uwe.assmann,jakob.henriksson}@tu-dresden.de

Abstract. The First International Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2008), affiliated with the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS2008), brought together researchers and practitioners from the modeling community with experience or interest in MDE and in Knowledge Representation to discuss about: (1) how the scientific and technical results around ontologies, ontology languages and their corresponding reasoning technologies can be used fruitfully in MDE; (2) the role of ontologies in supporting model transformation; (3) and how ontologies can improve designing domain specific languages.

1 Introduction

As Model Driven Engineering spreads, disciplines like model transformation and domain specific modeling become essential in order to support different kinds of models in an model driven environment. Understanding the role of ontology technologies like knowledge representation, automated reasoning, dynamic classification and consistence checking in these fields is crucial to leverage the development of such disciplines.

Thus, the objectives of the First International Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2008) were to present success cases of integrated approaches and state-of-the-art researches covering ontologies in MDE; and to encourage the modeling community to explore different aspects of ontologies.

TWOMDE 2008 addressed how the scientific and technical results around ontologies, ontology languages and their corresponding reasoning technologies can be used fruitfully in MDE. More specifically, TWOMDE 2008 discussed the infrastructure for integrating ontologies and MDE and the application of ontologies in the following aspects of MDE: Domain Specific Languages, Design Patterns, and Conceptual Modeling.

This first edition counted on one invited talk and five paper presentations. The audience comprised 20 participants. Senior researchers and professors constitute at least half of audience. It indicates that the modeling community is willing to know about the integration of Ontologies and MDE.

This report covers an analysis of papers presented in the workshop. For more information, please refer to the Workshop Proceedings[1].

2 Research Papers

The workshop was divided in three parts. It started with the invited talk about about "potential applications of ontologies and reasoning for modeling and software engineering" following by a group of papers concerning application of ontologies in MDE.

Andreas Friesen gave a keynote talk about the experience of SAP's potential applications for ontologies and reasoning for enterprise applications [1]. Participating of at least five EU projects on the topic, SAP has collected a large number of potential applications. We illustrate two of them: Dynamic Integration of logistic service providers and business process composition.

The first involves the usage of semantic web services and ontologies to automatically find the most appropriate web service based on predefined requirements. This application replaces multiple manual steps for discovery and selection of suitable web services.

The second potential application relies on employing ontologies in business process composition. When composing business processes, currently, there is a manual effort in ensuring the correct message flow and integration logic among business processes. Applying ontologies may allow for semi-automatic generating the message flow for consistent execution.

An open issue is how to measure the value added by ontologies. Indeed, although the role of ontologies is clear, metrics to assess the impact of ontologies on enterprise systems lack so far. Ongoing EU projects like MOST[2] may contribute with use cases and patterns to support this issue.

2.1 Applications of Ontologies in MDE

Papers addressing the application of ontologies in MDE cover topics like design pattern integration, domain specific languages and multi-agent systems.

Cédric Bouhours presented the use of "an ontology to suggest design patterns integration" [2]. the paper analyses the application of an extended Design Pattern Intent Ontology (DPIO) in an pattern integration process. The process is

[1] `http://ceur-ws.org/Vol-395/`

[2] `www.most-project.eu`

composed by three steps: Alternative models detection, Validation of the propositions and Patterns integration. The DPIO ontology is used in the second step to validate the suggestions made. A future work would be the semi-automatic detection of alternative models by ontology. This task would make use of reasoning to infer relationships between the the model and the alternative model.

Another interesting application of ontologies is in the Domain Analysis of Domain-Specific Languages[3], presented by Marjan Mernik. In such paper, ontologies are used during the initial phase of domain analysis in identifying common and variable elements of the domain that should be modeled in a language for that domain. Since the research is on its first steps, the analysis of applying ontologies in the other stages was not considered yet. Currently, ontologies are applied in the domain analysis and automated reasoning services have not been used. Indeed, reasoning services could be used to build a class diagram from the ontology. For example, the common subsummer [4] can be used to suggest an abstract super class based on the description of two or more concrete subclasses.

Daniel Okouya [5] presented a paper with the proposal of applying ontologies in conceptual modeling of multi-agent systems (MAS) and uses the expressive power of OWL based ontologies to deal with constraints verification and domain knowledge provision of MAS models. The idea is to support designers providing verification and validation of conceptual models produced during the MAS development process.

2.2 Integrated Approaches

Marion Murzek presented an infra-structure for integrating ontologies in MDE in the paper "Bringing Ontology Awareness into Model Driven Engineering Platforms" [6]. The architecture is based on the authors' experience with interoperability issues in metamodeling platforms. It should provide support to the following MDE disciplines: (1) modeling, (2) management and (3) guidance.

For example, the framework supports applying ontologies to validating models (1), simulations and model transformations (2) and Flexibility of process definitions(3). This is an ongoing research with first prototypes scheduled for the second semester of 2009.

An approach from a different point of view was presented by Guillaume Hillairet in the paper "MDE for publishing Data on the Semantic Web" [7]. It proposes the usage of the object model as pivot between persistence layer and ontology in semantic web applications. Mapping and transformations between the object model and an ontology are discussed. An interesting conclusion is that MDE helps to reduce the complexity of dealing with these mappings and transformations.

3 Conclusion

The TWOMDE2008 was the first workshop at the MDE conference to address the application of ontologies in model driven development. The potential of this field has just started being explored.

Although we had papers covering different aspects of MDE, the employment of automated reasoning services to make use of the formal description provided by ontology languages has practically not been explored. Moreover, prominent topics in MDE like model transformation, traceability and query languages were not pondered by the papers of this first edition.

For the next editions, we expect more use cases in a wider range of topics. We would like to see successful industry use cases and mechanisms to evaluate the role of ontologies.

References

1. Friesen, A.: Potential applications of ontologies and reasoning for modeling and software engineering. In: Proceedings of the of the First Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2008) at MoDELS 2008, Toulouse, France, September 28, 2008. CEUR Workshop Proceedings, vol. 395 (2008) CEUR-WS.org
2. Harb, D., Bouhours, C., Leblanc, H.: Using an ontology to suggest design patterns integration. In: Proceedings of the of the First Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2008) at MoDELS 2008, Toulouse, France, September 28, 2008. CEUR Workshop Proceedings, vol. 395 (2008) CEUR-WS.org
3. Tairas, R., Mernik, M., Gray, J.: Using ontologies in the domain analysis of domain-specific languages. In: Proceedings of the of the First Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2008) at MoDELS 2008, Toulouse, France, September 28, 2008. CEUR Workshop Proceedings, vol. 395 (2008) CEUR-WS.org
4. Mantay, T.: Computing least common subsumers in expressive description logics. Technical report, Hamburg, Germany (1999)
5. Okouya, D., Penserini, L., Saudrais, S., Staikopoulos, A., Dignum, V., Clarke, S.: Designing mas organisation through an integrated mda/ontology approach. In: Proceedings of the of the First Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2008) at MoDELS 2008, Toulouse, France, September 28, 2008. CEUR Workshop Proceedings, vol. 395 (2008) CEUR-WS.org
6. Zivkovic, S., Murzek, M., Kuehn, H.: Bringing ontology awareness into model driven engineering platforms. In: Proceedings of the of the First Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2008) at MoDELS 2008, Toulouse, France, September 28, 2008. CEUR Workshop Proceedings, vol. 395 (2008) CEUR-WS.org
7. Hillairet, G., Bertrand, F., Lafaye, J.Y.: Mde for publishing data on the semantic web. In: Proceedings of the of the First Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2008) at MoDELS 2008, Toulouse, France, September 28, 2008. CEUR Workshop Proceedings, vol. 395 (2008) CEUR-WS.org

Using an Ontology to Suggest Software Design Patterns Integration

Dania Harb, Cédric Bouhours, and Hervé Leblanc

IRIT – MACAO
Université Paul Sabatier
118 Route de Narbonne
F-31062 TOULOUSE CEDEX 9
{harb,bouhours,leblanc}@irit.fr

Abstract. To give a consistent and more valuable feature on models, we propose that model-driven processes should be able to reuse the expert knowledge generally expressed in terms of patterns. In order to formalize and use them, some design pattern ontologies have been developed. To share them on the Web they have been implemented using the OWL language. They can be easily interrogated with dedicated query languages. Our work has consisted in extending a design pattern intent ontology with "alternative model" and "strong points" concepts, which partially refers "anti-patterns". We validate this approach in tooling a step of a design review activity, we have proposed. This activity, directed by design patterns, is adapted to a model driven process, for the need to improve object-oriented architecture quality.

Keywords: OWL, SPARQL, Software Design Pattern, Design Review Activity, MDE, MDA.

1 Introduction

The emergent MDE community, aiming at giving a productive feature on models, has proposed model-driven process development. However, to obtain guarantees on model relevance at the end of each activity, these processes should promote the reuse of the knowledge of experts generally expressed in terms of analysis [1], design [2] or architectural [3] patterns approved by the community. Given the existence of "code review" activities [4] in some development processes, we have specified a "design review" activity [5] directed by design patterns and oriented to model quality. In this activity, we propose to parse models to find fragments substitutable with software design patterns and to replace them if the intent of the designer matches with the intent of the pattern and if the architectural qualities of the pattern are needed. Our activity is situated after the design stage, and its purpose is to urge and to help the designer to integrate design pattern in his design.

Thanks to their *Design Pattern Intent Ontology* (DPIO), Kampffmeyer et al. [6] have developed a wizard enabling designers to efficiently retrieve software design patterns applicable for their design problems, during the design stage. Our approach has not the same timing. It is situated after the design stage, and it verifies if there is no known bad design practices in a model. So, the designer is not in need of identifying design problems, it is the activity which finds the lacks in his models and suggests design patterns integrations instead. However, the DPIO [6] is an interesting start point for the formalization of our concepts because it links software design pattern to some problem concepts. So, in reusing this ontology backwards (from the pattern to the design problems), and in adding our concepts, we are able to establish a dialog with the designer.

In this paper, after presenting the design review activity, we explain how we have reused and extended the DPIO [6]. We illustrate the execution of our activity on a “file system management” example.

2 The Design Review Activity

The design review activity, presented in [5], may be decomposed into four steps (see Fig. 1).

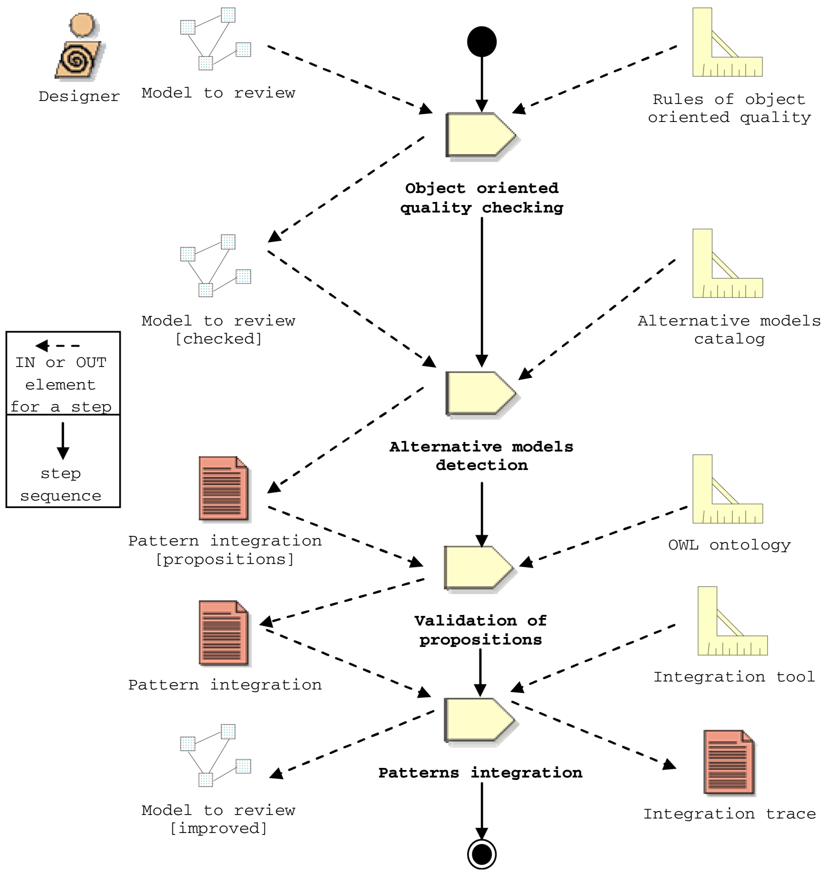

Designer
Model to review
Rules of object oriented quality
Object oriented quality checking
Model to review [checked]
Alternative models catalog
IN or OUT element for a step
step sequence
Alternative models detection
Pattern integration [propositions]
OWL ontology
Validation of propositions
Pattern integration
Integration tool
Patterns integration
Model to review [improved]
Integration trace

Fig. 1. Design Review Activity

In order to work with models in a “sufficient” quality, the first step checks good basic object-oriented design practices.

When the model to review is checked in a “sufficient” quality state, the second step consists in an automatic research of model fragments which are candidate to a substitution with a pattern. This research is based on structural similarities detection with “alternative models”. An “alternative model” is a model which solves inadequately the same problem as a pattern [5]. That means there is a better solution to solve this problem. Our work hypothesis is that a software design pattern is the best solution for a given design problem. According to the taxonomy proposed by Chikofsky and Cross [8], our detection technique can be connected to a redocumentation technique as to permit model restructuring. Our “alternative models” catalog is presented in [9], with the experiments used to constitute it.

Each “alternative model” detected in the model represents propositions of fragments substitutable with a design pattern. Since we need the designer opinion in the third step, our ontology will help him determine if his intent matches with the suggested pattern intent and whether the propositions are needed in the model to review.

With the designer authorization, the last step consists in integrating the validated propositions into the model. This integration is done thanks to an automatic model refactoring.

3 Reusing and Extending an Existing Ontology

In order to improve the design of object oriented models, our work relies on detecting all instances of alternative models in UML design models and substituting them, if necessary, with appropriate design patterns. Each class of the instances detected is connected in the same manner as the classes of the alternative model. So, since the detection is only structural, the instances detected must be checked by the designer before any substitution with a pattern. Therefore, after the detection step, propositions of patterns integration consist of sets of model fragments representing a possible substitution. These sets may be large where some fragments may not be relevant with a substitution. So, to help the designer in filtering the fragments, we need an ontology that formalizes intent of design patterns (is the substitution have a sense?) and our characterizations of “alternative models” in terms of quality features (is the effort of the substitution balanced by improved architectural qualities?).

For this purpose, we choose OWL, the Web Ontology Language [10], to import an existing ontology on design patterns intent and extend it by adding our knowledge on “alternative models”. We validated our new knowledge base using a specific query language to interrogate it and filter out the pertinent information.

3.1 Requirements

Our catalogue is composed with “alternative models”, introduced in Section 2, and their characterization. We have constituted our catalog in asking students to solve some design problems. These problems were simply solvable with designs patterns, but, as the students chosen have no knowledge on design patterns, they solve the problems without using design patterns. In following our work hypothesis, their solutions were not the best solution for the problem, and so, the models produced had some design defects. The characterization of these defects consists in a valuation of the "strong points" of the pattern. "Strong points" are criteria of object-oriented architecture or software engineering quality, partially deduced from the "consequences" section of the GoF [2] catalogue and from our study on the design defects of "alternative models". As pattern injection may alter some object-oriented metrics [11], "strong points" allow us to compute dedicated pattern metrics to classify the "alternative models" and to help the estimation of the pertinence of pattern injection in a design model. Each "alternative model" perturbs the "strong points" of its associated pattern.

Since we need to formally describe design patterns, "alternative models" and "strong points" in a machine readable format, we start with the DPIO ontology. These concepts must be constructed in a way that allows querying based on the "alternative model" detected.

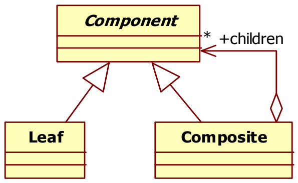
In Fig. 2, we present one of the GoF patterns, named *Composite*. The intent, the applicability and the structure are provided directly from the GoF book while the "strong points" are deduced from our experiments by comparing solutions to specific design problem implemented by the *Composite* pattern and its "alternative models". Fig. 3 shows the structure and the design defect valuation of an "alternative model" to the *Composite* pattern. We have named it "Development of the composition on Composite with process conformance" in reference of its design defects. Then an "alternative model" can be considered as a "chipped pattern".

Intent: Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Applicability: Use the Composite pattern when:

- you want to represent part-whole hierarchies of objects.
- you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.

Structure:



Strong points:

1 Decoupling and extensibility
  1.1 Maximal factorization of the composition
  1.2 Addition or removal of a Leaf does not need code modification
  1.3 Addition or removal of a Composite does not need code modification
2 Uniform processing
  2.1 Uniform processing on operations of composed object
  2.2 Uniform processing on composition managing
  2.3 Unique access point for the client

Fig. 2. Composite Pattern and its "Strong Points"

Name:

Development of the composition on *Composite* with process conformance

Alternative model:

Strong points perturbations			
1.1	✗	2.1	✓
1.2	✗	2.2	✓
1.3	✗	2.3	✓

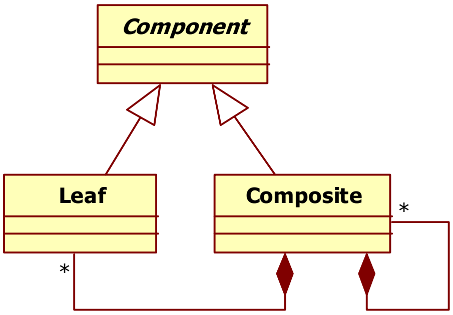


Fig. 3. Characterization of an "Alternative Model"

So we have made two major hypotheses about "alternative models". First, each "alternative model" is attached by the valuation of their design defects to a unique design pattern. Second, each "alternative model" has one or more strong points perturbed. We assume that the same structure of an "alternative model" can be duplicated in our catalog, but with a different name, a different valuation and some different components.

3.2 Existing Ontology: The Design Pattern Intent Ontology

Design patterns have been used successfully in recent years in the software engineering community in order to share knowledge about the structural and behavioural properties of software. Most of the existing approaches to formalizing design patterns are based on structural aspects. For example, the work of Dietrich et al. [12] uses the OWL to formally describe the structure of design patterns and then transform it in first-order logic predicates which are reuse as an entry for a scanner pattern. However, there is more lightly approaches concentrated in the usability of design patterns according to the design problems they solve. Kampffmeyer and Zschaler [6] define the intent of the 23 GoF design patterns [2] using OWL. Their work was based on the work of Tichy [13], who developed a catalogue of more than hundred design patterns classified according to the problems patterns solve.

The core structure of the DPIO, provided from the paper [6], is presented in Fig. 4 by UML classes and associations. Kampffmeyer and Zschaler chose to represent their ontology with UML diagram because they consider that is easily to understand. To read the diagram, they indicate: "The relations between *DesignPattern*, *DPProblem* and *ProblemConcept* classes are depicted using UML-notations. UML classes symbolize OWL classes and UML associations symbolize OWL object properties. Each *DesignPattern* is a solution to one or more design pattern problem *DPProblem*. The association between them indicates an object property *isSolutionTo* which is an inverse property of *isSolvedBy*. *DPProblem* is defined that is the root node for more specific problems. The association class *Constrains* indicates an OWL object property

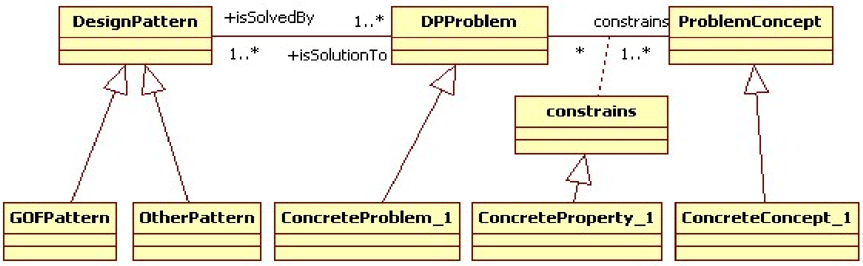


Fig. 4. Graphical overview of the core structure of the DPIO

that can be specialized also by subproperties. *DPProblem* is a set of classes that describe a problem by constraining a *ProblemConcept"*. The DPIO contains the vocabulary for describing the intent of design patterns.

All the 23 GoF patterns inherit from the DesignPattern class. DPProblem and ProblemConcept are the root classes of the other hierarchies.

Based on the work of [6], and instead of finding the design pattern for a given problem, we retrieve the intent of a design pattern. It is much like reversing the query to get the pertinent data from the same ontology. So we can benefit from their existing work and their published ontology.

3.3 Method and Results

Now to determine the scope of our new ontology, there are kinds of questions called "competency questions" the ontology should be able to answer [14]. Our work could be defined in 3 steps: first, when an "alternative model" is detected, we need to interrogate our knowledge base to know which design pattern could replace it. Second, we will verify with the designer if his "alternative model" detected has a similar intent as the corresponding design pattern. Last, in this case, we will show him the lack in his model by displaying the perturbed "strong points". Then, if the designer finds the need to improve his model, his fragment will be substituted with the design pattern. Therefore, the three competency questions are as follow:

1. Which design pattern could replace a given "alternative model"?
2. What is the intent of the corresponding design pattern?
3. Which are the "strong points" perturbed using this "alternative model"?

In designing the structure of the new ontology, we took into consideration all the possible relations between the classes in the DPIO model and the classes we want to add:

1. Each"*alternative model"* could be replaced by one and only one *Design Pattern*. But a *Design Pattern* will replace one to many "alternative models".
2. An "*alternative model*" *perturbs* at least one "s*trong point"* of the *Design Pattern* that can replace it.

From this analysis, we extend the DPIO by adding our new concepts.

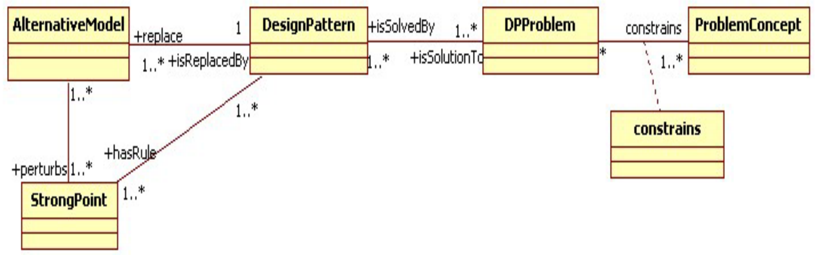


Fig. 5. Graphical overview of the structure of the extended ontology

Fig. 5 represents the new structure of the extended ontology. Based on this structure and the relations between classes, we extended the existing ontology with OWL classes and properties as follow:

1. Two new OWL classes:
   a. *AlternativeModel*: the root class of all "alternative models". They are grouped by the design pattern that could replace them. For example, we find six "alternative models" for the *Composite* pattern. They inherit all from the *Composite_AM* class (Fig. 6). They have the name of their super class followed by their numeration in the catalogue.
   b. *StrongPoint*: the root class of all the "strong points". They are attached to a design pattern. For example, we find two main "strong points" for the *Composite* pattern: *Composite_Rule_1* and *Composite_Rule_2* (Fig. 6); each one of them was précised by three sub features. They have the name of their super class followed by their numeration in the catalogue.

2. Four new OWL properties:
   a. *isReplacedBy*: links an *AlternativeModel* to his corresponding *DesignPattern*.
   b. *Replace*: the inverse of *isReplacedBy*.
   c. *Perturbes*: links an *AlternativeModel* to the valuation of the corresponding pattern "strong points" (*StrongPoint*).
   d. *hasRule*: links a *DesignPattern class* to one of its *StrongPoint.*

Fig. 6 shows a detailed structure of the extended base concerning the *Composite* pattern. The "alternative model" presented in Fig. 3 perturbs the three subfeatures of the first "strong point" of the *Composite* pattern that concerned in the *Decoupling and extensibility*. More precisely, for each OWL class concerning our concepts, we have:

OWL Classes	***rdfs:comment***
Composite_AM_5	Development of the composition on "Composite" with protocol conformance
Composite_Rule_1	Decoupling and Extensibility
Composite_Rule_2	Uniform processing
Composite_Rule_1.1	Maximal factorization of the composition

Composite_Rule_1.2	Adding or removing a Leaf does not need a code modification
Composite_Rule_1.3	Adding or removing a Composite does not need a code modification
Composite_Rule_2.1	Uniform processing on operations of composed objects
Composite_Rule_2.2	Uniform processing on compositions management
Composite_Rule_2.3	Unique access point for the client

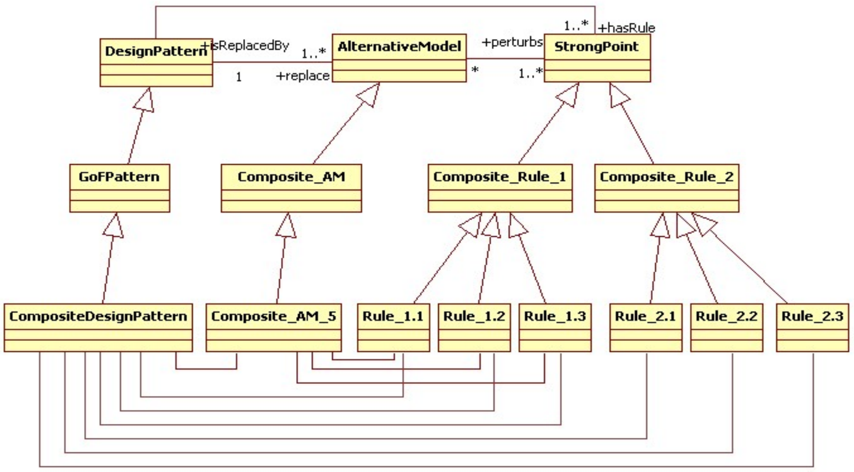


Fig. 6. Detailed structure of the extended ontology

For presentation reasons, we have omitted the name of the design pattern in each sub feature.

We used Protégé [15], an open source ontology editor and knowledge-base framework, to load the existing ontology and add our new classes, properties, property characteristics, and interrogate it using queries. We referred to a user guide [14] on how to develop an ontology using Protégé and the OWL Plug-in. We created our OWL classes, linked them by OWL properties, and interrogated the knowledge base by generating SPARQL (SPARQL Protocol and RDF Query Language) [16] queries to answer our competency questions.

SPARQL is a W3C Candidate Recommendation towards a standard query language for the Semantic Web. Its focus is on querying RDF graphs at the triple level. *SPARQL* can be used to query an RDF Schema or OWL model to filter out individuals with specific characteristics.

4 Illustration on a "File System Management" Design

After adding our new concepts to the DPIO, the knowledge base could now be interrogated according to the competency questions we mentioned earlier. Standard ontology reasoning is used to retrieve the results responding to queries. In order to illustrate the use of the ontology, we execute the whole activity on an example. It was found in a subject of an object-oriented programming supervised practical work. It aims to implement a file management system represented in the Fig. 7 below.

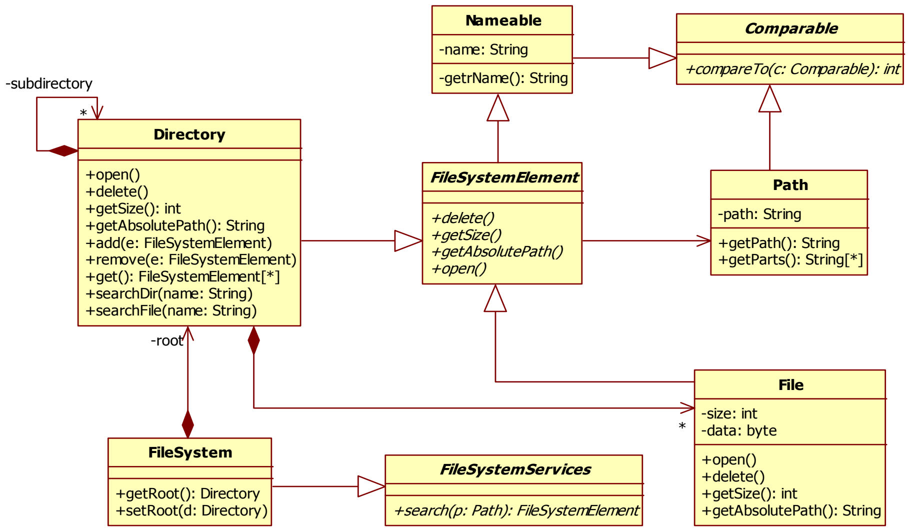


Fig. 7. Model to Review: File System Management

This static UML model represents a basic architecture for a File System Management. Authors of this model are interested in the presentation of some object concepts:

- Inheritance between classes and abstract classes. A uniform protocol for every *FileSystemElement* is encapsulated by a corresponding abstract class. *Directories* and *Files* must respect this protocol via inheritance relationship. We can note that all concrete classes are derived directly or indirectly from an abstract class. This rule enforces the emergence of reusable protocols.
- Management of references, here composition links, between container and components. A *Directory* object manages some references to *Files* and *Directories* objects.

Nevertheless, this model contains a misconception. Although there is a uniform protocol owned by the class *FileSystemElement*, the management of composite links along a hierarchical structure is duplicated. Indeed, *Directory* class manages independently links on *Files* and *Directories*. Now, we consider two evolution scenarios. The first is adding new Terminal types in the tree structure, for example, symbolic links in UNIX systems. This evolution requires the management of this new type of links by the *Directory* class and then requires code modification and code duplication in this class. The second is adding new non Terminal types in the tree structure, for example archive files in UNIX or in Java environment. We can consider that an archive file has the same functionalities as a *Directory*. This evolution requires a reflexive link on an archive file class and the duplication of all links that represent composition links in the tree structure. Then it requires duplication of management of composition and modification in the *Directory* class, it must manage another type on *FileSystemElement*. These two scenarios show a decoupling problem (each container manages a part of the composite structure) and an extensibility limitation (it requires existing code modification for adding new type of terminal or non terminal element of the composition hierarchy). Therefore, this model can be improved. Furthermore, when the authors have implemented this model, they realized that there were defects, and they adapted their code to improve it.

4.1 Object-Oriented Quality Checking

Visually, there is no design mistake: each class of the model presents a reusable protocol. Composition links are used here as delegation between *Directory* and *File*. And messages sent between them have the same selector.

4.2 "Alternative Models" Detection

This step consists in the execution of all queries corresponding at each "alternative model" of the base. In this example, the query of the fifth *Composite* "alternative model" returns theses matching classes:

1. The *Directory* class is able to play the role of the *Composite* class.
2. The *File* class is able to play the role of the *Leaf* Class.
3. The *FileSystemElement* is able to play the role of the *Component* class.

This means that we detected an "alternative model" for the *Composite* pattern because they have the same structural features (cf. Fig. 8).

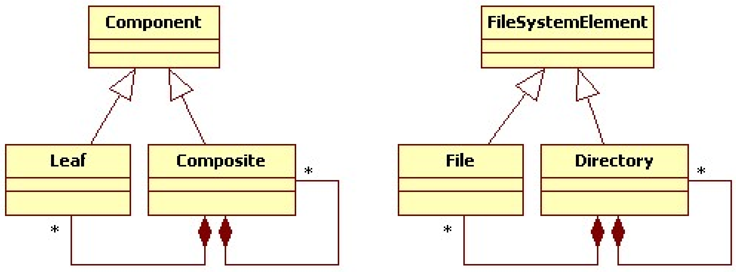


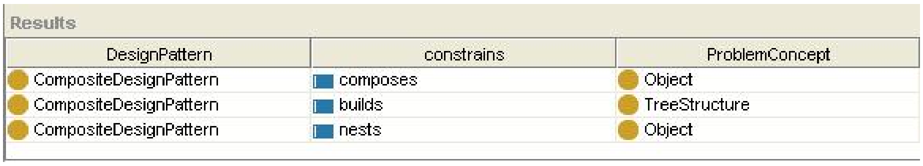
Fig. 8. The fifth *Composite* "Alternative Model" its Instantiation in the Model

4.3 Designer/Machine Dialog

At this step, the designer must verify the substitutability of the detected fragment. Firstly, he must verify if the intent of the fragment matches with the proposed design pattern. To do so, we build a question thanks to a SPARQL query we have coded (cf. Listing 1). This query retrieves the intent of the design pattern in using the "alternative model" detected (here *Composite_AM_5*). Indeed, we consider that the intent of the pattern is described with a list of couples (constraint – ProblemConcept) in the ontology (see Fig. 5).

```
SELECT ?DesignPattern ?constrains ?ProblemConcept
WHERE{
         ?DesignPattern rdfs:subClassOf ?x.
         ?x rdf:type owl:Restriction.
         ?x owl:onProperty :replace.
         ?x owl:someValuesFrom: Composite_AM_5.
         ?DesignPattern rdfs:subClassOf ?y.
         ?y rdf:type owl:Restriction.
         ?y owl:onProperty :isSolutionTo.
         ?y owl:someValuesFrom ?pbconcept.
         ?pbconcept rdfs:subClassOf ?z.
         ?z rdf:type owl:Restriction.
         ?z owl:onProperty ?constrains.
         ?z owl:someValuesFrom ?ProblemConcept.
}
```

Listing 1. SPARQL query to retrieve the intent of the *Composite* pattern that could replace the "alternative model" *Composite_AM_5*


Results

DesignPattern	constrains	ProblemConcept
● CompositeDesignPattern	▬ composes	● Object
● CompositeDesignPattern	▬ builds	● TreeStructure
● CompositeDesignPattern	▬ nests	● Object

Fig. 9. Screenshot of Protégé after executing the query (Listing 1)

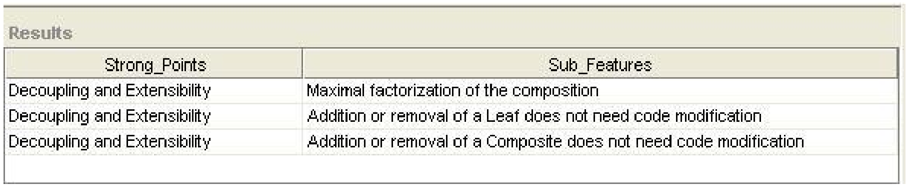
Based on the results (cf. Fig. 9) of this query, we will proceed in dialoguing the designer with the first question: *We have detected in your design an alternative model of the CompositeDesignPattern. Is the fragment {FileSystemElement, File, Directory} composes Object, builds TreeStructure and nests Objects?*

We can note that the intent of {*FileSystemElement, File, Directory*} is a recursive composition: "*Directories* are composed with *Files* or *Directories* which are composed with…". So the answer to the previous question is positive.

Now, we must check the interest to replace the fragment with the pattern. Thanks to the perturbation of the "strong points", we can present to the designer the advantage to use the pattern. We retrieve the perturbed "strong points" with a SPARQL query (Listing 2):

```
SELECT ?Strong_Points ?Sub_Features
WHERE{
         :Composite_AM_5 rdfs:subClassOf ?x.
         ?x rdf:type owl:Restriction.
         ?x owl:onProperty :perturbs.
         ?x owl:someValuesFrom ?SF.
         ?SF rdfs:subClassOf ?SP.
         ?SP rdfs:comment ?Strong_Points.
         ?SF rdfs:comment ?Sub_Features.
} ORDER BY ASC(?Strong_Points)
```

Listing 2. SPARQL query to retrieve the "strong points" perturbed by *COMPOSITE_AM_5*



Results

Strong_Points	Sub_Features
Decoupling and Extensibility	Maximal factorization of the composition
Decoupling and Extensibility	Addition or removal of a Leaf does not need code modification
Decoupling and Extensibility	Addition or removal of a Composite does not need code modification

Fig. 10. Screenshot of the result window presenting the "strong points" perturbed

The second question is built with the results (cf. Fig. 10) of the previous query: *Our analysis shows that you have problems of "Decoupling and Extensibility"; your model is unable to satisfy those points:*

1. *Maximal factorization of the composition.*
2. *Addition or removal of a leaf does not need code modification.*
3. *Addition or removal of a composite does not need code modification.*

In injecting the CompositeDesignPattern, you will improve all of these points. Do you want to refactor the identified fragment {FileSystemElement, File, Directory} ?

As we consider that the model may evolve, it is useful to guarantee that there are extensibility and decoupling possibilities. Therefore, the fragment must be substituted with the pattern.

4.4 Patterns Integration

At this step, the identified fragment is replaced by the suggested design pattern like the Fig. 11 below:

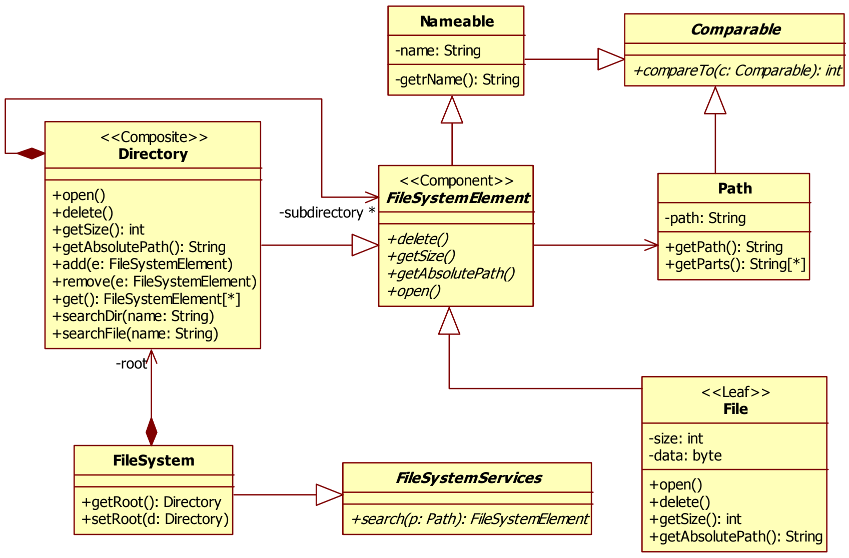


Fig. 11. Model to Review Improved

To do so, a suite of simple model refactoring suffices to integrate the pattern. Here, it consists in:

- Remove composition link between Directory and File.
- Move the end of the recursive composition link from Directory to FileSystemElement.

These inter-classes refactorings can be automatically deduced with an operation of “differentiation” between the “alternative model” and the pattern structure.

At the end of the activity, we can say that this model is improved, because we have substituted a fragment (with “weak points”) with a pattern (with “strong points”). This transformation may appear as non fundamental in the model, but at the code level, the implications are substantial. Every hierarchy traversal methods are simpler to implement, and there is less code to write. Moreover, in case of extensions, there is no code modification of existing classes.

5 Conclusion and Perspectives

The approach of reusing and extending an existing ontology corresponding to our requirements was successfully applied. From the existing DPIO ontology, we have plugged our concepts on “alternative models” and “strong points”. These concepts are fundamental for tooling our Design Review Activity. Accurately, at the step named validation of substitution propositions, we have simulated a dialog with a designer by interrogating the extended base using queries. These queries will be generated automatically by a template process. The integration of this work into a tool dedicated to the design review activity is envisaged.

Finally, we conclude with some perspectives:

- Take into consideration the relationships between patterns. For example, the *Decorator* pattern can be applied to the *Composite* pattern structure.
- Take into consideration the applicability of each pattern. For example, referring to the GoF book, one of the applicability of the Composite pattern is: *you want clients to be able to ignore the difference between compositions of objects and individual objects*. We notice that this sentence cannot be part of the pattern intention but can be considered as a “strong point”.
- Optimize our knowledge base by sharing common “strong points” between patterns. For example, the Composite, the Decorator and the Bridge pattern have a same “strong point” concerning the maximal factorization between specific classes.
- Use inference rules to find new concepts when adding new “alternative models” or “strong points”. This could help us improving our knowledge on patterns and particularly, our knowledge on the good practices on object oriented architecture.

Acknowledgements

We are grateful to Mrs. Nathalie Aussenac-Gilles for her precious advices during this work.

References

1. Fowler, M.: Analysis patterns: reusable objects models. Addison Wesley Longman Publishing Co, Inc., Amsterdam (1997)
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley Professional, Reading (1995)
3. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture. John Wiley & Sons, Chichester (1996)
4. Dunsmore, A.P.: Comprehension and Visualisation of Object-Oriented code for Inspections. Technical Report, EFoCS-33-98, Computer Science Department, University of Strathclyde (1998)
5. Bouhours, C., Leblanc, H., Percebois, C.: Alternative Models for a Design Review Activity. In: Kuzniarz, L., Sourrouille, J.-L., Staron, M. (eds.) Workshop on Quality in Modeling - ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, Nashville, TN, USA, 30/09/2007-05/10/2007, pp. 65–79. Springer, Heidelberg (2007)
6. Kampffmeyer, H., Zschaler, S., Engels, G., Opdyke, B., Schmidt, D.C., Weil, F.: Finding the pattern you need: The design pattern intent ontology. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 211–225. Springer, Heidelberg (2007)
7. Guéhéneuc, Y.G., Albin-Amiot, H.: Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-Class Design Defects. In: Proceedings conference TOOLS, pp. 296–305 (July 2001)
8. Chikofsky, E.J., Cross, J.H.: Reverse engineering and design recovery: A taxonomy. IEEE Software 7(1), 13–17 (1990)
9. Bouhours, C., Leblanc, H., Percebois, C.: Alternative Models for Structural Design Patterns, research report, IRIT/RR–2007-1–FR, IRIT (December 2007), `http://www.irit.fr/recherches/DCL/MACAO/docs/AlternativeModelsForStructuralDesignPatterns.pdf`
10. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview (2004), `http://www.w3c.org/TR/owl-features/`
11. Huston, B.: The effects of design pattern application on metric scores. Journal of Systems and Software 58(3), 261–269 (2001)
12. Dietrich, J., Elgar, C.: A formal description of design patterns using OWL. In: Australian Software Engineering Conference (ASWEC 2005), pp. 243–250. IEEE Computer Society, Los Alamitos (2005), `http://doi.ieeecomputersociety.org/10.1109/ASWEC.2005.6`
13. Tichy, W.F.: A catalogue of general-purpose software design patterns. In: TOOLS 1997. Proceedings of the Tools-23: Technology of Object-Oriented Languages and Systems. IEEE Computer Society, Washington (1997)
14. Noy, N.F., McGuinness, D.L.: Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05, Knowledge Systems Laboratory, Stanford University, Stanford, CA, 94305, USA (March 2001)
15. Protégé ontology editor and knowledge acquisition system (2006), `http://protege.stanford.edu/`
16. Prud'hommeaux, E.: Seaborne: SPARQL Query Language for RDF (January 2008), `http://www.w3.org/TR/rdf-sparql-query/`

Using Ontologies in the Domain Analysis of Domain-Specific Languages

Robert Tairas[1], Marjan Mernik[2], and Jeff Gray[1]

[1] University of Alabama at Birmingham, Birmingham, Alabama, USA
{tairasr,gray}@cis.uab.edu
[2] University of Maribor, Maribor, Slovenia
marjan.mernik@uni-mb.si

Abstract. The design stage of domain-specific language development, which includes domain analysis, has not received as much attention compared to the subsequent stage of language implementation. This paper investigates the use of ontology in domain analysis for the development of a domain-specific language. The standard process of ontology development is investigated as an aid to determine the pertinent information regarding the domain (e.g., the conceptualization of the domain and the common and variable elements of the domain) that should be modeled in a language for the domain. Our observations suggest that ontology assists in the initial phase of domain understanding and can be combined with further formal domain analysis methods during the development of a domain-specific language.

Keywords: Domain Analysis, Domain-Specific Languages, Ontology.

1 Introduction

The development of a Domain-Specific Language (DSL) requires detailed knowledge of the domain in which the language is being targeted. Paradigms such as *Generative Programming* [3] and *Domain Engineering* [5] also require an understanding of the target domain, which is done through a process called domain analysis that produces a domain model. An important theme in the domain analysis used by both paradigms is the need to determine elements that can be reused. The reusable components or software artifacts form the building blocks for developing new software systems. In DSL development, in addition to the overall knowledge of the domain, the domain model can reveal important properties that will influence the way the language is shaped. In particular, the search for reusability in domain analysis can be translated into realizing the *commonalities* and *variabilities* of a domain. This information can assist in pointing out elements in the domain that can be fixed in the language and those that must provide for variabilities; hence, domain analysis has the potential to be beneficial if used during DSL development. However, clear guidelines for the use of established domain analysis techniques in the process of DSL development are still lacking [11].

Ontology development is one approach that has contributed to the early stages of domain analysis [5]. This paper investigates the use of ontology during domain analysis in DSL development and how it contributes to the design of the language. The rest of the paper is organized as follows: Section 2 describes the potential connection between ontology and DSL development. Section 3 provides a case study on the use of ontology in the development of a DSL for air traffic communication and Section 4 provides some observations on ontology in DSL development based on the case study. Related work, a conclusion, and future work are described in Sections 5 and 6.

2 Early Stage DSL Development

Chandrasekaran et al. [2] propose two properties related to ontologies: the first is a representation vocabulary of some specialized domain. This vocabulary represents the objects, concepts, and other entities concerning the domain. The second is the body of knowledge of the domain using this representative vocabulary. This knowledge can be obtained from the relationships of the entities that have been represented by the vocabulary. Ontologies seek to represent the elements of a domain through a vocabulary and relationships between these elements in order to provide some type of knowledge of the domain.

An interesting connection can be observed between ontology and DSL design. As it relates to DSL development [11], a domain model is defined as consisting of:

- a domain definition defining the scope of the domain,
- the domain terminology (vocabulary, ontology),
- descriptions of domain concepts, and
- feature models describing the commonalities and variabilities of domain concepts and their interdependencies.

Not only is an ontology useful in the obvious property of domain terminology, but the concepts of the domain and their interdependencies or relationships are also part of the properties of an ontology [2]. The knowledge of the commonalities and variabilities of the domain concepts can further provide crucial information needed to determine the fixed and variable parts of the language. This part is a more open question as to the potential of finding commonalities and variabilities through information obtained from the ontology.

As it relates to the DSL development process as a whole, the insertion of ontology development in the early stages of DSL development can potentially provide a structured mechanism in the part of DSL development that is still lacking attention. The early stages of DSL development (i.e., domain analysis) have not received as much attention compared to the latter stages of development (i.e., language implementation). Various DSL implementation techniques have been identified in [11], including interpreter or compiler development and embedding in a General-Purpose Language (GPL). In contrast, only four out of 39 DSLs evaluated in [11] utilized a more formal domain analysis, such as FAST [14] and FODA [8]. These formal approaches have shown to result in good language design, but their use is still limited and it has yet to be seen how well they will be adopted by the community. The question is whether other less formal approaches, such as Object-Oriented Analysis (OOA) or ontology, can be reused in the early stages of DSL development. In order to promote interest in the domain analysis stage of DSL development, this paper advocates the use of ontology in DSL development, which is observed through a case study of a DSL for air traffic communication.

3 Case Study

Ontology development to assist in the design of a DSL is described through a case study in this section. Section 3.1 provides a summary of the air traffic communication problem domain. The ontology and its related competency questions are given in Sections 3.2 and 3.3. The development of a class diagram, object diagram, context-free grammar, and sample program related to the DSL and obtained from the ontology is given in Section 3.4.

3.1 Air Traffic Communication

A case study was selected to apply the ontology development process and observe its usefulness in domain analysis related to DSL development. The case study selected focuses on the communication that occurs between the air traffic control (ATC) at an airport and the pilot of an aircraft. More specifically, the communication is between the ground controller that is responsible for the traffic between the runways and the ramps containing gates in an airport, and the pilots of an aircraft that has just arrived or is in the process of departure. The purpose is to develop a DSL that can standardize the language for the communication between the two individuals. English is the standard language in this domain, but more often the controllers or pilots of non-English speaking countries may experience problems communicating in English. A DSL that standardizes the communication can be translated into the native tongue of the controller or pilot for better comprehension. A separate functionality could check and verify the path that is given to a pilot by a ground controller. An example communication sequence that highlights the potential communication problem is given in Listing 1. The controller is asking the captain to hold short of taxiway "MikeAlpha," but the pilot continually assumes it is taxiway "November."

Listing 1. Example of air traffic communication

ATC:	Make the right turn here at Juliette. Join Alpha. Hold short *MikeAlpha*.
Pilot:	Right on Juliette hold sh ... Taxi Alpha. Hold *November* [...] Can we taxi now?
ATC:	Make the right turn here at Juliette. Join Alpha. Hold short of *MikeAlpha*.
Pilot:	Roger, join right Juliette. Join Alpha. Hold short *November*.
ATC:	OK, I'll say it again. Hold short of *Mike Alpha* "M" - "A" *MikeAlpha*, not *November*.
Pilot:	OK, hold short of *MikeAlpha*.

3.2 Ontology Development

Following the ontology development process outlined by Noy and McGuinness [13], competency questions are selected that serve as the purpose of the ontology.

In order to obtain a domain model as defined in Section 2, two competency questions were selected: “What are the concepts of the domain and the interdependencies of these concepts?” and “What are the commonalities and variabilities of the domain?”

Both the Ontolingua[1] and DAML[2] ontology libraries were searched for existing ontologies related to the domain in this case study, but no related instances contained the vocabulary necessary for the domain. Although a new ontology is needed for this case study, the availability of an existing ontology in other cases provides a head start to the development of a domain model as the important terms and relationships have been determined already for the domain and can be used toward the subsequent steps of DSL development.

Table 1. Listing of classes and associated slots

Class	Description	Slots		
		Name	Description	Values
`Aircraft`	Arriving or departing aircraft	`Airline ID`	Name of the airline	Two letters
		`Flight Number`	Flight Identification	Integer
`GroundControl`	Controller in charge of airport ground traffic			
`Tower`	Controller in charge of take-offs and landings			
`Runway`	Available take-off and landing locations	`Runway Number`	Runway Identification	1 – 36 (i.e., runway heading 10° – 360°)
		`Runway Orientation`	To distinguish parallel runways	Class `Left` or `Right`
`Taxiway`	Paths connecting runways to ramps	`Taxiway Name`	Taxiway Identification	One or two letters (digits)
`Ramp`	Aircraft parking area	`Ramp Name`	Ramp Identification	String
`Gate`	Passenger embarkation and disembarkation	`Gate Letter`	Gate Identification	One letter
		`Gate Number`	Gate Identification	Integer
`Turn`	Command to turn	`Direction`	Turning direction	Class `Left` or `Right`
		`Taxiway`	Taxiway Identification	Class `Taxiway`
`HoldShort`	Command to hold short of a runway or taxiway	`Runway`	Runway Identification	Class `Runway`
		`Taxiway`	Taxiway Identification	Class `Taxiway`
`Contact`	Command to contact a separate controller	`ATC`	Controller to contact	Class `Tower` or `GroundControl`
`Follow`	Command to follow behind another aircraft	`Aircraft`	Aircraft Identification	Class `Aircraft`

[1] Ontolingua Ontology Library, http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html
[2] DAML Ontology Library, http://www.daml.org/ontologies

Utilizing the tool introduced by Noy and McGuinness [13] called Protégé 2000[3], the ontology for the case study was developed. The terms in Protégé 2000 are stored as classes. This allows for terms to be considered subclasses of other terms. In addition to classes, Protégé 2000 also contains slots and instances. Slots are the properties and constraints of the classes. Slots define the properties of classes and also determine the values that can be set for each property. Instances are actual instances of the classes in the ontology. These can be used to determine how well the ontology is representing a domain.

Table 1 contains a selection of classes and slots of the ontology that was developed in Protégé 2000 for the case study. In addition to the classes and slots in Table 1, instances of these classes were also determined. These instances are based on the information from a simplified diagram of the Birmingham International Airport (BHM) as shown in Figure 1. For example, instances of the `Runway` class are 6, 24, 18, and 36. Instances of the `Taxiway` class are A, B, F, G, H, M, A1, A2, A3, A4, B1, G1, H2, and H4. The `Ramp` class consists of Cargo and Terminal.

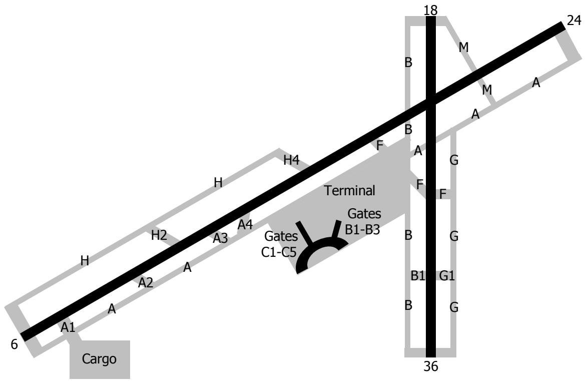


Fig. 1. Simplified diagram of Birmingham International Airport (BHM)

3.3 Competency Questions Revisited

The usefulness of the ontology in Table 1 can be measured by how well the ontology assists in answering the previously specified competency questions from Section 3.2. Regarding the first question, the ontology provides the concepts of the domain through the classes. The interdependencies between the concepts can be derived from the constraints of the slots of the classes. For example, the `HoldShort` class is dependent on either the `Runway` or `Taxiway` classes, as this command is always followed by the location in which the pilot is to hold short.

Answering the second question related to commonalities and variabilities is less evident if observing only the ontology's structure of classes and slots. Information regarding the variabilities can be extracted by including the instances of classes, such as the instances from BHM. Classes `Runway` and `Taxiway` consist of many instances, which could mean these classes have the variabilities property. Moreover, instances that represent airports other than BHM will also contain different values for these

[3] Protégé 2000, http://protege.stanford.edu

classes, which could also be interpreted as containing variabilities. The classes not containing instances, such as most of the commands (i.e., `Turn`, `HoldShort`, and `Contact`), could be interpreted as common concepts in all instances. These commands are common in the ATC domain and represent standard commands that are used in all airports. However, the specific airport elements (i.e., collection of instances of runways and taxiways) may change depending on the airport.

3.4 Conceptual Class Diagram

The ontology process is similar to the process of object-oriented analysis [1]. However, one distinction is that ontology design is mainly concerned with the *structural* properties of a class, whereas object-oriented analysis is primarily concerned with the *operational* properties of a class [13]. The focus here is a methodology that can assist in determining the domain concepts for DSL development by reusing an approach from general software engineering.

Figure 2 presents a conceptual class diagram that was manually generated from the structural information of the classes in the ontology from Table 1. In this case, the development of the class diagram has been assisted by the information obtained from the ontology. In Figure 2, similar classes are grouped together. For example, classes `Gate`, `Ramp`, `Runway`, and `Taxiway` represent physical structures in the airport. Such groupings identified the need for a generalized class for each group. A generalized class was included in the diagram for `Runway` and `Taxiway`, because from the slot properties of class `HoldShort`, two possible values can be used (i.e., `Runway` and `Taxiway`). In the diagram, this is represented by abstract class `Way`. The classes at the bottom of the diagram represent communication commands. These are associated with other classes through their

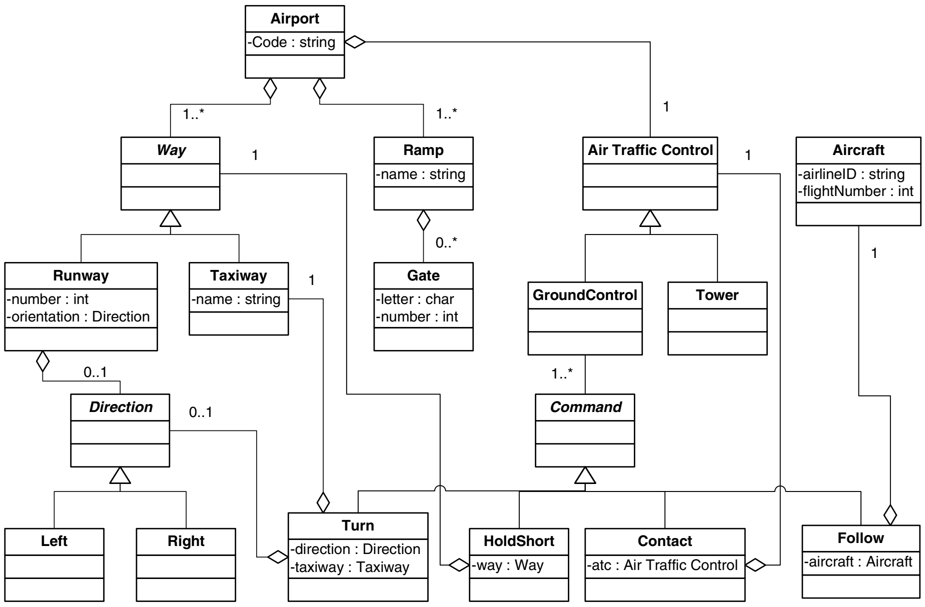
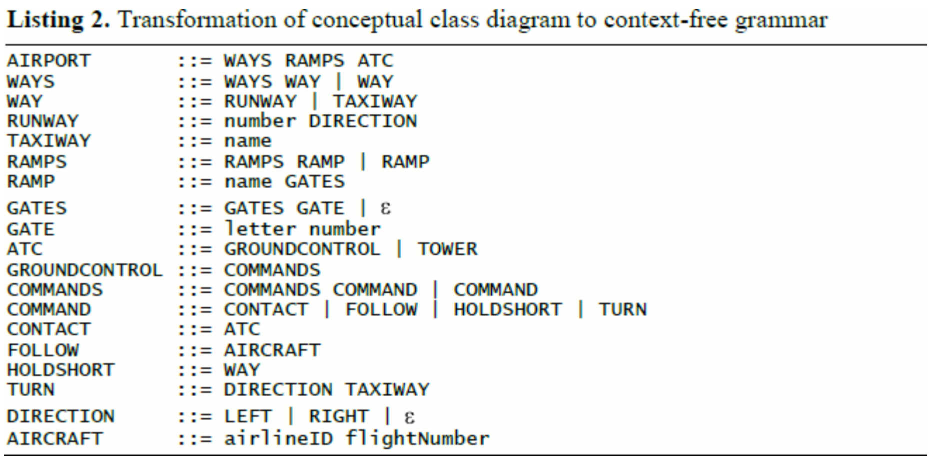


Fig. 2. Conceptual class diagram obtained from the ontology

respective slot properties. Generalizations such as `Command` and `Way` were not part of the original ontology and were only introduced during the development of the class diagram. These classes in turn can be used to update the ontology to further improve the structure of the ontology. This can be seen as part of the process of iteratively refining the ontology to better represent the domain.

From the class diagram in Figure 2, an initial context-free grammar (CFG) for the DSL can be generated, as shown in Listing 2. This CFG was manually obtained from the conceptual class diagram to CFG transformation properties defined in [12]. Relationships such as *generalization* and *aggregation* in the class diagram are transformed into specific types of production rules in the CFG. For example, a generalization where classes `Runway` and `Taxiway` are based on class `Way` is transformed into the production rule `WAY ::= RUNWAY | TAXIWAY`. An aggregation where class `Gate` is part of class `Ramp` is transformed into the production rule `RAMP ::= GATES`. In this case the non-terminal `GATES` is used, because the cardinality of this aggregation is zero or more gates on a ramp (i.e., 0..*). An additional production rule is generated to represent this cardinality (i.e., `GATES ::= GATES GATE | ε`).

Listing 2. Transformation of conceptual class diagram to context-free grammar


Listing 2. Transformation of conceptual class diagram to context-free grammar

```
AIRPORT        ::= WAYS RAMPS ATC
WAYS           ::= WAYS WAY | WAY
WAY            ::= RUNWAY | TAXIWAY
RUNWAY         ::= number DIRECTION
TAXIWAY        ::= name
RAMPS          ::= RAMPS RAMP | RAMP
RAMP           ::= name GATES
GATES          ::= GATES GATE | ε
GATE           ::= letter number
ATC            ::= GROUNDCONTROL | TOWER
GROUNDCONTROL  ::= COMMANDS
COMMANDS       ::= COMMANDS COMMAND | COMMAND
COMMAND        ::= CONTACT | FOLLOW | HOLDSHORT | TURN
CONTACT        ::= ATC
FOLLOW         ::= AIRCRAFT
HOLDSHORT      ::= WAY
TURN           ::= DIRECTION TAXIWAY
DIRECTION      ::= LEFT | RIGHT | ε
AIRCRAFT       ::= airlineID flightNumber
```

The transformation of the class diagram into the CFG above, albeit manual, followed a predefined collection of transformation rules. The manual transformation of the ontology into the class diagram is less formal, but was done by connecting the properties of the classes in the ontology with the graphical representation of the class diagram. In order to provide a more automated transformation between the ontology and the class diagram, developing a transformation between a Web Ontology Language (OWL) instance for the ontology and a textual representation of the class diagram could be considered. Related to this, UML-based ontology development has been proposed [6]. Specifically for this case, the transformation between an XML-based OWL file into a class diagram represented in XMI could assist in the automation of the ontology to class diagram step. After the transformation to a CFG, some keywords have been added to the CFG for easier human parsing, as shown in Listing 3.

Listing 3. Addition of keywords and production refactoring

```
AIRPORT       ::= WAYS RAMPS ATC
WAYS          ::= WAYS WAY | WAY
WAY           ::= runway RUNWAY | taxiway TAXIWAY
RUNWAY        ::= number DIRECTION
TAXIWAY       ::= name
RAMPS         ::= RAMPS RAMP | RAMP
RAMP          ::= ramp name GATES
GATES         ::= GATES GATE | ε
GATE          ::= gate letter number
ATC           ::= GROUNDCONTROL | TOWER
GROUNDCONTROL ::= COMMANDS
COMMANDS      ::= COMMANDS COMMAND | COMMAND
COMMAND       ::= CONTACT | FOLLOW | HOLDSHORT | TURN
CONTACT       ::= contact ATC
FOLLOW        ::= follow AIRCRAFT
HOLDSHORT     ::= hold short WAY
TURN          ::= turn DIRECTION on TAXIWAY
DIRECTION     ::= left | right | ε
AIRCRAFT      ::= airlineID flightNumber
TOWER         ::= tower
```

An example of a program written in this DSL is shown in Listing 4 and is based on the CFG of Listing 3. Even from this simple DSL for ground control, it can be seen that some simple verification of aircraft path control can be checked. The development of the DSL has been aided by the ontology that was initially produced, which in turn assisted in the generation of a class diagram. This provided a means to understand the domain in the early stages of DSL development, which provided input to the subsequent structure of the DSL, as seen in the grammar in Listing 2.

Listing 4. An example program

```
// description of BHM airport
runway 6 runway 24 runway 18 runway 36
taxiway A taxiway A1 taxiway A2 taxiway A3 taxiway A4 taxiway B  taxiway B1
taxiway F taxiway G  taxiway G1 taxiway H  taxiway H2 taxiway H4
ramp Cargo
ramp Terminal gate B1 gate B2 gate B3 gate C1 gate C2 gate C3 gate C4 gate C5

// commands from Ground Control
turn right on A
turn left on M
hold short runway 18
contact tower
```

An object diagram of the example program in Listing 4 is illustrated in Figure 3. Airport-related structures such as gates, ramps, runways, and taxiways are represented by multiple objects that will differ among various airports. However, the types of commands issued by the ground control remain the same. The specific attributes of the command objects are based on the objects of the structures of a particular airport, e.g., taxiway A and M, and runway 18. As described in Section 3.3, evaluating the instances of the classes provides information regarding the elements of the domain that are common (or fixed) and those that are variable.

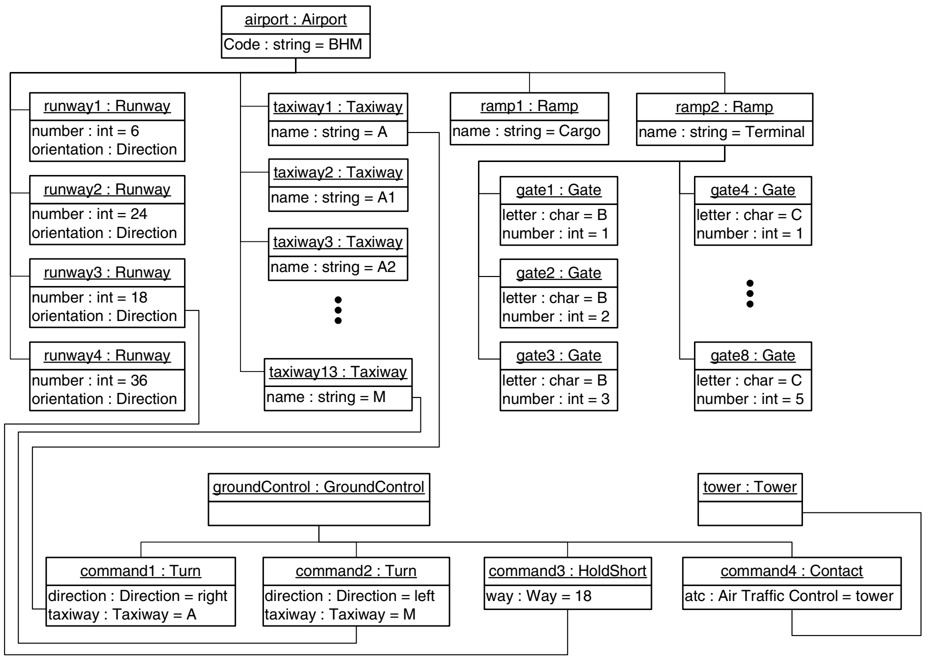


Fig. 3. Object diagram from example program

4 Ontologies in DSL Development

Section 3 summarized the development of a preliminary ontology using the standard development process as seen in literature using a well-known tool called Protégé 2000. The usefulness of the ontology was measured by answering several competency questions that were selected to match the goals of domain analysis. Domain concepts and their interdependencies were determined. In addition, commonalities and variabilities as they relate to the DSL can be determined by observing the instances of the classes in the ontology. It should be noted that the ontology and class diagram went through several iterations before reaching the state described in Section 3. However, further refinements may help to provide more satisfactory answers to the competency questions. The ontology was then used to manually generate a conceptual class diagram, which in turn produced an initial context-free grammar for the proposed DSL.

The case study has shown the potential usefulness of ontology in the development of a DSL specifically during the early stages of development. An ontology can provide a well-defined and structured process to determine the concepts of a domain and the commonalities and variabilities for a DSL, which can result in the generation of a class diagram and subsequently a CFG from the information. Two further observations highlight the benefits of an ontology-based approach. First, if an ontology is already available for a domain, then this existing ontology can be used to initiate the development of a DSL without the need to start from scratch. This was not the case for the air traffic communication domain described in Section 3, but ontologies for other domains could already exist and be utilized in the DSL development for those domains. Second, the entire process outlined in Section 3 could be used as an alternative to a more formal domain analysis technique such as FODA. In a separate direction, the ontology alone can be combined with formal domain analysis techniques (e.g., proposed by Mauw et al. in [10]) and be used as a supplier of a well-defined input of domain concepts and relationships for further analysis.

5 Related Work

De Almeida Falbo et al. describe the use of ontology in domain engineering that has the purpose of developing software artifacts for *reuse* [5]. A more recent publication demonstrates the use of ontology in engineering design requirements capture [4]. Both cases propose methodologies of utilizing ontology in terms of providing the knowledge about a specific domain, albeit more in a general case of engineering. However, the utilization of ontology in domain analysis in these works translates well to the similar effort in DSL development. Guizzardi et al. associate ontology with the development of Domain-Specific Visual Languages (DSVL) [7]. The ontology is used to assist in developing a representative language for a specific domain that is correct and appropriate. Similarly, our initial investigation described in this paper utilizes ontology as part of the main goal of developing a representative language for a domain such as air traffic communication. However, in addition to this, the common and variable elements of the domain are also considered through the ontology in order to determine the structure of the domain-specific textual language (i.e., fixed and variable language constructs).

Gašević et al. describe efforts to associate the two technical spaces of Model-Driven Architecture (MDA) and ontology, which include the utilization of MDA-based UML in ontology development [6]. We follow a similar approach where a connection is made between the ontology in Table 1 and the UML class diagram in Figure 1. However, in addition to this association, we perform manual transformations on the class diagram to obtain a context-free grammar for the DSL.

6 Conclusion and Future Work

An initial investigation of the usefulness of ontology in domain analysis in DSL development was described in this paper. A case study demonstrated the insertion of ontology development in the DSL development process, where a class diagram was obtained from the ontology and subsequently a CFG was produced. The ontology assisted in answering questions related to the domain, such as the main concepts and their interdependencies, and the common and variable parts related to the DSL. The ontology also provided a structured input to the subsequent stages of DSL development. Continued exploration of ontology-driven domain analysis may provide further proof of effectiveness in the analysis of domains for DSL development.

The class diagram in Figure 2 that was generated from the ontology can also serve as the basis for creating a metamodel. Slight adaptations of this diagram could represent the metamodel for a tool like the Generic Modeling Environment (GME) [9], which provides a domain-specific modeling language that has a concrete syntax that resembles concepts from the domain. Thus, the results of the domain analysis and the observed ontology can inform technologies of both grammarware and modelware. This direction will be explored as future work. In addition, the transformations that were performed were done manually based on predefined transformation properties. A possibility for a more automated step is the transformation of the Web Ontology Language (OWL) representation into a Backus-Naur Form (BNF) representation for the DSL. Such a transformation may map similar elements and perform some alterations between the representations. This direction will also be considered in future work.

Acknowledgments. This project is supported in part by NSF grant CPA-0702764.

References

[1] Booch, G.: Object-Oriented Development. IEEE Transactions on Software Engineering 12, 211–221 (1986)

[2] Chandrasekaran, B., Josephson, J., Benjamins, V.: What Are Ontologies, and Why Do We Need Them? IEEE Intelligent Systems 14, 20–26 (1999)

[3] Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Boston (2000)

[4] Darlington, M., Culley, S.: Investigating Ontology Development for Engineering Design Support. Advanced Engineering Informatics 22, 112–134 (2008)

[5] De Almeida Falbo, R., Guizzardi, G., Duarte, K.: An Ontological Approach to Domain Engineering. In: International Conference on Software Engineering and Knowledge Engineering (SEKE), Ischia, Italy, pp. 351–358 (2002)

[6] Gašević, D., Djurić, D., Devedžić, V.: Model Driven Architecture and Ontology Development. Springer, Berlin (2006)

[7] Guizzardi, G., Ferreira Pires, L., van Sinderen, M.: Ontology-Based Evaluation and Design of Domain-Specific Visual Modeling Languages. In: International Conference on Information Systems Development, Karlstad, Sweden (2005)

[8] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)

[9] Lédeczi, Á., Bakay, Á., Maróti, M., Völgyesi, P., Nordstrom, G., Sprinkle, J., Karsai, G.: Composing Domain-Specific Design Environments. IEEE Computer 34, 44–51 (2001)

[10] Mauw, S., Wiersma, W., Willemse, T.: Language-Driven System Design. International Journal of Software Engineering and Knowledge Engineering 14, 625–664 (2004)

[11] Mernik, M., Heering, J., Sloane, A.: When and How to Develop Domain-Specific Languages. ACM Computing Surveys 37, 316–344 (2005)

[12] Mernik, M., Črepinšek, M., Kosar, T., Rebernak, D., Žumer, V.: Grammar-Based Systems: Definition and Examples. Informatica 28, 245–255 (2004)

[13] Noy, N., McGuinness, D.: Ontology Development 101: A Guide to Creating Your First Ontology, `http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf`

[14] Weiss, D., Lay, C.: Software Product Line Engineering. Addison-Wesley, Boston (1999)

Model-Driven Development of Context-Aware Web Applications Based on a Web Service Context Management Architecture

Georgia M. Kapitsaki and Iakovos S. Venieris

National Technical University of Athens,
School of Electrical and Computer Engineering,
Intelligent Communications and Broadband Networks Laboratory,
Heroon Polytechniou 9, 15773 Athens, Greece
gkapi@icbnet.ntua.gr, venieris@cs.ntua.gr

Abstract. Context information constitutes an essential aspect of service development and provision in mobile computing in the attempt to provide users with personalized services. The problem of handling context in these environments, as well as the development of context-aware services, have become quite challenging research tasks in the last years. In this paper, the ongoing work towards context handling of web services is presented along with a model-driven methodology for context-aware service engineering for web applications built on web services. The solution focuses on decoupling the context management mechanism from the core service logic in all development stages.

1 Introduction

An increasing trend towards mobile computing is visible in the last years. The main aim of application developers for these environments is to offer users personalized services that take into account location, personal information and current activity, as well as other contextual information. Under this perspective, there is a vital need to manage the use of contextual information in mobile applications. At the same time, there is an increasing trend towards offering Internet services to mobile users and this tends to be performed through stripped down version of accepted web services that have been tested in the Internet world. The development of context-aware services, i.e. services that take into account context information and adapt their behaviour accordingly, is an interesting research challenge in the field of software engineering that needs to be handled in all development changes, since in many occasions the delivery of a service requires a complicated software engineering process passing through the stages of analysis and design prior to the actual code development. When web services are exploited for the service development the development process should be treated accordingly.

In this paper a solution to the above problem is proposed by the research work towards the model-driven development of context-aware web applications consisting of web services. The combination with model-driven techniques helps in addressing context management in all development stages. An approach that focuses mainly on the decoupling of the main service logic from the context management phase is presented. Indeed, an interesting observation regarding the context adaptation of user centric services is that although the interaction between the user and the service can be adapted to contextual parameters, the overall goals related to service logic are usually indifferent to context changes. As an example we can consider a service providing airport ticket bookings. Depending on the user context (e.g. financial constraints, preferred airlines) different operations belonging possibly to different subservices can be invoked. In every case the main service goal - ticket booking - remains the same regardless of the specific user context. Along this line of thought, core service logic and context handling can be treated as two separate concerns, both on modeling and implementation level. Finally, not all application cases depend on context, but context adaptation may or may not be applied depending on service characteristics. For this reason, the context adaptation scheme needs to remain transparent to the service and be applied on top of it.

The proposed work shows that a structured methodology including service and context modeling along with the corresponding dependencies is sufficient to result in a functional web application built on web services, where context information is handled in all service development stages remaining outside the main application logic. By doing so it contributes to the advance of the research on model-driven development of context-aware services by providing a methodology that handles context information independently in all phases and uses concrete transformation tools and code mappings without limiting to the specification of the development meta-models. Indeed descriptions of concrete code generation tools is something that is not addressed in detail in the activities of this research field. Of course the approach is targeting a specific area of interest - which has not been studied extensively in the past - regarding the use of web services to form the context-aware web application. Nevertheless, by exploiting the proposed ideas the application consisting of web service clients can be generated from the model allowing the developer to potentially reuse existing services and focus on the dependencies with context information. Furthermore, the developer does not need to be fully aware of how the context adaptation mechanism works during execution allowing him again to concentrate primarily on the application design.

The rest of the paper is structured as follows. Section 2 introduces the main concepts of context and context-awareness used in the paper concentrating also on related work in the field, whereas Section 3 presents the proposed model-driven methodology along with the specified steps and the introduced profiles. Section 4 is dedicated to the context-management architecture on which the methodology is based. Section 5 presented a simple modeling example that demonstrates how the proposed methodology can be used and finally Section 6 concludes the paper and outlines the current and future work in respect to the presented approach.

2 Context Awareness and Context-Aware Services

Context information can be defined in many ways and in different domains. In the domain of mobile applications and services the most widely used definition has been given by Day and Abowd stating that: *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves"* [1]. Many different solutions have been provided by researchers to the problem of context management during service development and provision. Regarding the provision dedicated platforms or middleware are trying to provide supporting infrastructures like in the cases of the Service-oriented Context-Aware Middleware (SOCAM) [2] and the Context-awareness sub-structure (CASS) [3].

In service development a variety of solutions have also been proposed. In many cases the context management is handled directly at the code level by enriching the business logic of the service with code fragments responsible for performing the context manipulation. The first such significant attempt can be found in Context Oriented Programming (COP) [4] where the code skeleton is kept context-free and context-dependant stubs are exploited for context manipulation. Context-filling is then used for selecting the appropriate stub and binding it with the respective program gap, thus providing the desired context-dependent behaviour. Another example are context-aware aspects [5] that exploit the principles of Aspect-oriented Programming (AOP) [6] to separate context-aware behaviour from application code. Aspects are context-aware in the sense that their use is driven by context. A certain aspect may or may not be executed depending on its context of use.

However, context adaptation needs to be handled in the preceding development stages especially when developers face more complicated cases. Model-driven engineering and Model Driven Architecture (MDA) [7] techniques have been exploited by many researchers (e.g. in [8]), in order to meet this need. Some of the proposed solutions are focusing on modeling using Domain Specific Modeling (DSM) like [9], where the WebML CASE tool is used. Context is rendered here as a first-class actor allowing applications to adapt automatically to context changes without user intervention targeting context-aware multichannel web applications. Although specific to other types of applications than the ones assumed in our work, this approach focuses more on information modeling for web applications, whereas in the proposed approach the functionality of the application is addressed in addition. In the majority of cases though meta-models that extend the widely used Unified Modeling Language (UML) [10] are proposed. Meta-modeling extends the abstract UML syntax to allow its specialized use in different domains of context. Examples of this latter category are the UML meta-model of [11] that supports structural, architectural and behavioural adaptations of service design based on context values and [12], where context is categorized in state-based context that characterizes the current situation of an entity and event-based context that represents changes in an entity's state. These approaches use generic service dependencies to context only at design level, whereas in the presented work focus is given on web service dependencies that can also be managed during service execution.

We are adopting the approach of the latter group that allows us to focus on the whole development life cycle. Indeed model-driven techniques provide an abstraction from the concrete platform and facilitate significantly the developer's work. However, they could prove inefficient without adequate support at the code level. Therefore the proposed model-driven approach is also oriented towards an implemented architecture responsible for the context adaptation of context-aware web services at runtime. The architecture is based on message interception for the context adaptation of the web services that form the application. Message interception approaches have also been proposed in the literature like the work presented in [13] where the context information transport and processing for web services is addressed. Context information is maintained in blocks and is included as extensions in message headers, whereas the context manipulation is automatically performed through handlers either included as context plugins in the framework or available remotely as dedicated web services.

3 Proposed Model-Driven Development Methodology

The proposed solution process towards context-aware development of web applications is based on an open solution that employees widely used languages and practices and is presented in Fig. 1. The methodology is initiated by the introduction of the separate service model (Step 1) and the context service model (Step 2) to be used in the application, all in UML notation. Service models refer to the services that act as business services for the application, i.e. the available web services that are to be adapted to context, whereas context service models refer to the web services that act as context sources and have access to the actual context information in different ways (e.g. by encapsulating the functionality provided by sensors, remote database systems etc.). The context associations between the context sources and the application context model - depicting which source is used to retrieve each context information - forms part of Step 3. Having both models and source assignments available the context dependencies between the service and context models are designed (Step 4). Then the application flow for the complete web application expressing the order of web service operations calls consisting of web service consumers is constructed for applications that consist of more than one web service (Step 5). Finally, the application is mapped to platform specific code (e.g. Java or C++ source code files, configuration files etc.) through the appropriate code transformation (Step 6).

For the design phase, a number of UML profiles are proposed: an intuitive web service profile and a context profile based on a modified version of the ContextUML metamodel [15] similar to the one presented in [16] depicted both in Fig. 2. The web service profile contains the elements found in a Web Services Description Language (WSDL) document, whereas for the context modeling it is assumed that all business services and context sources are exposed as web services.

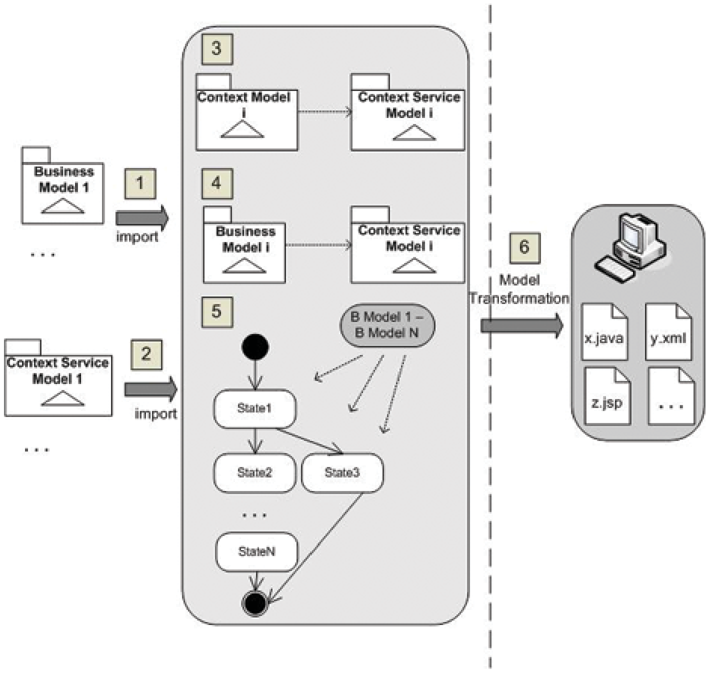


Fig. 1. The steps for the model-driven development of context-aware applications

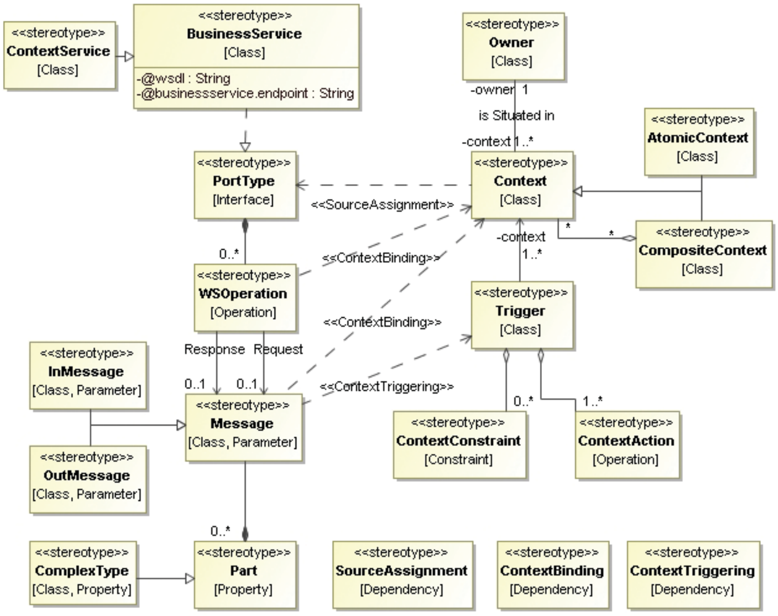

<<stereotype>> ContextService [Class]
<<stereotype>> BusinessService [Class]
-@wsdl : String
-@businessservice.endpoint : String
<<stereotype>> Owner [Class]
-owner 1
is Situated in
-context 1..*
<<stereotype>> AtomicContext [Class]
<<stereotype>> PortType [Interface]
<<SourceAssignment>>
<<stereotype>> Context [Class]
<<stereotype>> CompositeContext [Class]
-context 1..*
<<stereotype>> WSOperation [Operation]
<<ContextBinding>>
<<ContextBinding>>
<<stereotype>> Trigger [Class]
Response Request
<<ContextTriggering>>
<<stereotype>> InMessage [Class, Parameter]
<<stereotype>> Message [Class, Parameter]
<<stereotype>> OutMessage [Class, Parameter]
<<stereotype>> ContextConstraint [Constraint]
<<stereotype>> ContextAction [Operation]
<<stereotype>> ComplexType [Class, Property]
<<stereotype>> Part [Property]
<<stereotype>> SourceAssignment [Dependency]
<<stereotype>> ContextBinding [Dependency]
<<stereotype>> ContextTriggering [Dependency]

Fig. 2. The web service and context profiles

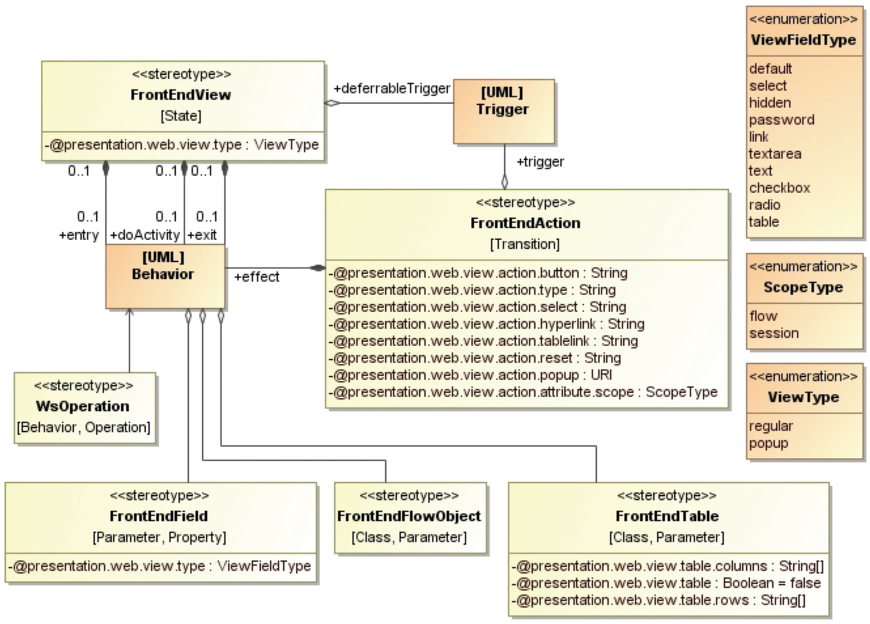
Building on the introduced profiles, the separate business and context service models are either imported in the modeling environment or constructed (Steps 1 and 2 mentioned above). They can even be generated based on the WSDL web service definitions. The source assignment for the context model using `<<SourceAssignment>>` stereotyped relationships that represent relations between context information and the web service source is then performed. This step is followed by the modeling of the relations between the two main models: the business logic and the context model, since the dependencies between the service and context model are also part of the profile definitions.

Three different cases of context adaptation are supported by the development methodology and the implemented architecture:

– *Parameter Injection*: one or more parameters values of the request message are replaced with values related to context information (e.g. a parameter representing a location is set to the current user location at the time of service invocation). This is performed using `<<ContextBinding>>` stereotyped associations and involves the modification of the `<<Part>>` parameters of the `<<InMessage>>` stereotyped classes.
– *Operation selection*: the operation contained in the request message is changed to reflect the contextual needs. This case is useful for services that aggregate a number of methods implementing the same functionality in different ways. The operation to be invoked is selected based on context information (e.g. a payment service may select the method that corresponds to the payment choice stored in a user profile like payment using a credit card or a PayPal account). Again the `<<ContextBinding>>` stereotype between the service definition and the context information is used to reflect this dependency case.
– *Response Manipulation*: it refers to the manipulation or the modification of service responses based on context and is often related with filtering or sorting operations based on the context information. An example would be the ordering of a book list response according to user preferences on book genre. `<<ContextTriggering>>` associations between the service output and the trigger class (`<<Trigger>>` stereotype) with the triggered operation(s) (`<<ContextAction>>` stereotype) show this dependency. The respective operation is called when specific conditions expressed in the Object Constraint Language (OCL) [14] are met (`<<ContextConstraint>>` stereotype).

The above cases are specific to web services and capture the modifications that can be performed on web service messages. Of course it is also possible to have combinations of the above when adapting a service (e.g. a parameter injection for the input parameters followed by a response manipulation for the response).

In case of a web application consisting of more than one web services the service flow is modeled by means of a state transition diagram, which is suitable for service flow modeling in this type of applications. In the state transition diagram state activities correspond to web service operation calls that may be



<<enumeration>> ViewFieldType
default
select
hidden
password
link
textarea
text
checkbox
radio
table
<<stereotype>> FrontEndView [State]
-@presentation.web.view.type : ViewType
+deferrableTrigger
[UML] Trigger
+trigger
0..1
+entry
+doActivity
+exit
[UML] Behavior
+effect
<<stereotype>> FrontEndAction [Transition]
-@presentation.web.view.action.button : String
-@presentation.web.view.action.type : String
-@presentation.web.view.action.select : String
-@presentation.web.view.action.hyperlink : String
-@presentation.web.view.action.tablelink : String
-@presentation.web.view.action.reset : String
-@presentation.web.view.action.popup : URI
-@presentation.web.view.action.attribute.scope : ScopeType
<<enumeration>> ScopeType
flow
session
<<enumeration>> ViewType
regular
popup
<<stereotype>> WsOperation [Behavior, Operation]
<<stereotype>> FrontEndField [Parameter, Property]
-@presentation.web.view.type : ViewFieldType
<<stereotype>> FrontEndFlowObject [Class, Parameter]
<<stereotype>> FrontEndTable [Class, Parameter]
-@presentation.web.view.table.columns : String[]
-@presentation.web.view.table : Boolean = false
-@presentation.web.view.table.rows : String[]

Fig. 3. The presentation profile

adapted to context if specified so by the application model. For the state diagram modeling an introduced Presentation profile is exploited. The profile is used to model application view properties, User Interface (UI) elements and the application navigation. At the code level the presentation UI flow is handled by the application controller that can be found in Model-View-Controller (MVC) based architectures (e.g. Spring Web Flow[1]). Generally, the presentation profile captures the properties of such MVC pattern implementations (e.g. Apache Struts[2] or Spring MVC[3]). The profile is illustrated in Fig. 3 as part of the UML state transition diagram definition. It is comprised of different stereotypes such as `<<FrontEndView>>` which refers to a separate application view and tags in the general format:

$$@\text{presentation.web.view.type} = \text{text/enumerationtype}$$

that express more specific properties of view elements (e.g. if a web service return parameter should be displayed as a text field, a table, a hyperlink etc.).

For the code mapping step a transformation tool consisting of a model parser and a code generator has been implemented. At the current state the implemented code generator targets the context adaptation architecture presented next. However, the mapping can also be performed considering target implementations based on different web development technologies, since the fundamental

[1] http://springframework.org/webflow

[2] http://struts.apache.org/

[3] http://springframework.org

web application development principles remain the same and the introduced profiles are generic enough to support different kind of code mappings. The aim is to extend the current work towards this direction. For the transformation process Eclipse EMF and UML2 libraries [17] along with a number of dedicated tools developed in the framework of this work are exploited. The presence of specific stereotypes on application model elements triggers the execution of Velocity templates[4] that are responsible for the generation of the actual source code (e.g. Java bean classes) and other configuration files (e.g. Spring web flow beans definition). The model is parsed in its EMF format which is exported from the XML Metadata Interchange (XMI) [18] representation. Different modeling environments can be exploited and many of them support this EMF export such as Magicdraw[5].

4 Proposed Solution to Context Management

In order to handle the context management problem at the platform specific level keeping it at the same time transparent to the core application logic, an architecture based on web services and the interception of Simple Object Access Protocol (SOAP) messages [19] is proposed (Fig. 4). The Apache Axis2[6] web service framework has been exploited for the implementation of the proposed mechanism, although the same principles can be applied for any framework that supports message interception. The context management is performed as follows: service requests and responses are intercepted through the architecture handler, context information related to these messages is retrieved and finally the modified messages reflecting the context adaptation are returned. The message modification is carried out through a number of plugins. The three adaptation cases presented earlier including combinations are supported by the management architecture.

The main elements of the architecture are:

– *SOAP Interception Handler*: the handler intercepts all SOAP request and response messages and loads the appropriate plugins for the message modification, if present. It is accompanied by a simple configuration file in XML format: *handler.xml* that lists the available plugins and associates context adaptation cases with the corresponding plugins.
– *Plugin library*: the plugins are responsible for the actual message modification. They are separated in inPlugins for SOAP requests and outPlugins for the responses. The context plugins communicate with the context sources that have direct access to the context information and alter the input and output messages to reflect the current user task and its environment. In this manner, the context adaptation is kept independent from the specific service implementation and clearly separated from the service logic. The plugins are

[4] http://velocity.apache.org/
[5] http://www.magicdraw.com/
[6] http://ws.apache.org/axis2/

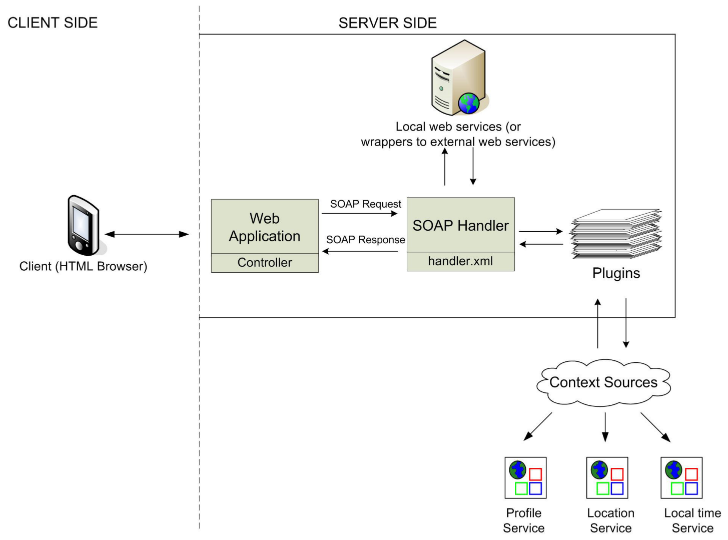


Fig. 4. The context management architecture

loaded on runtime based on a the configuration file. At the same time, the context adaptation phase remains optional, since no message modification is performed for services that do not require any adaptation.

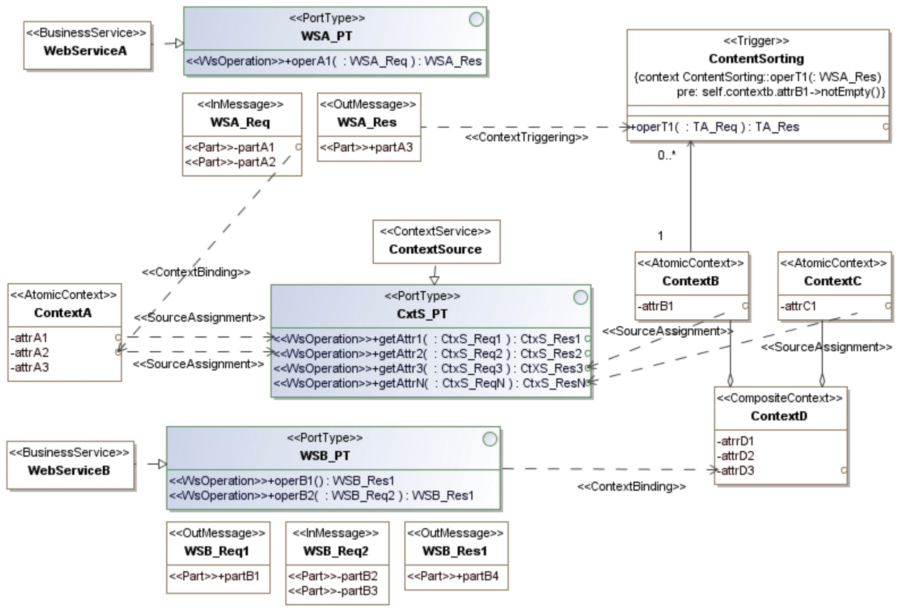
– *Context sources context providers*: the context information is accessed by the web services acting as context sources. The mechanisms that provide the information (e.g. sensors in the environment, RFIDs, database management systems etc.) expose their functionalities to these web services.

The architecture is presented extensively in [20].

5 Demonstration Example

In order to illustrate the above concepts a simple modeling example that follows the above principles is used. The example consists of two web services acting as business services `WebServiceA` and `WebServiceB`), one context source (`ContexSource`) and some atomic and composite context information (`ContextA` to `ContextD`). The service and context model with the corresponding dependencies are illustrated in Fig. 5.

All context adaptation cases are visible in the figure. The parameter injection case is visible between `partA1` of the input message of `WebServiceA` operation and the `attrA2` of `ContextA`. This means that the SOAP request message that will be intercepted by the handler needs to be modified. When this is done the value of the corresponding message part will be replaced by the value of the context information attribute which will be retrieved using the `getAttr2` context source operation as shown by the `<<SourceAssignment>>` dependency. The same operation is linked with a second dependency: a response manipulation



<<BusinessService>> WebServiceA
<<PortType>> WSA_PT
<<WsOperation>>+operA1(: WSA_Req) : WSA_Res
<<Trigger>> ContentSorting
{context ContentSorting::operT1(: WSA_Res) pre: self.contextb.attrB1->notEmpty()}
+operT1(: TA_Req) : TA_Res
<<InMessage>> WSA_Req
<<Part>>-partA1
<<Part>>-partA2
<<OutMessage>> WSA_Res
<<Part>>+partA3
<<ContextTriggering>>
0..*
1
<<ContextService>> ContextSource
<<ContextBinding>>
<<AtomicContext>> ContextB
-attrB1
<<AtomicContext>> ContextC
-attrC1
<<AtomicContext>> ContextA
-attrA1
-attrA2
-attrA3
<<SourceAssignment>>
<<PortType>> CxtS_PT
<<WsOperation>>+getAttr1(: CtxS_Req1) : CtxS_Res1
<<WsOperation>>+getAttr2(: CtxS_Req2) : CtxS_Res2
<<WsOperation>>+getAttr3(: CtxS_Req3) : CtXS_Res3
<<WsOperation>>+getAttrN(: CtxS_ReqN) : CtxS_ResN
<<SourceAssignment>>
<<SourceAssignment>>
<<SourceAssignment>>
<<CompositeContext>> ContextD
-atrrD1
-attrD2
-attrD3
<<BusinessService>> WebServiceB
<<PortType>> WSB_PT
<<WsOperation>>+operB1() : WSB_Res1
<<WsOperation>>+operB2(: WSB_Req2) : WSB_Res1
<<ContextBinding>>
<<OutMessage>> WSB_Req1
<<Part>>+partB1
<<InMessage>> WSB_Req2
<<Part>>-partB2
<<Part>>-partB3
<<OutMessage>> WSB_Res1
<<Part>>+partB4

Fig. 5. Application modeling example service and context models diagram

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <plugins>
    <plugin id="1" pckg="plugins.wsa" classname="WSAReqInPlugin"
            jarfile="wsareqpl.jar"/>
    <plugin id="2" pckg="plugins.wsa" classname="WSAResOutPlugin"
            jarfile="wsarespl.jar"/>
    <plugin id="3" pckg="plugins.wsb" classname="WSBReqInPlugin"
            jarfile="wsbreqpl.jar"/>
    <association>
        <serviceEnd name="WebServiceA" operation="operA1"/>
        <pluginEnd id="1" direction="in"/>
    </association>
    <association>
        <serviceEnd name="WebServiceA" operation="operA1"/>
        <pluginEnd id="2" direction="out"/>
    </association>
    <association>
        <serviceEnd name="WebServiceB" operation="operB1"/>
        <pluginEnd id="3" direction="in"/>
    </association>
</plugins>
```

Fig. 6. Handler.xml configuration file for the presented dependencies

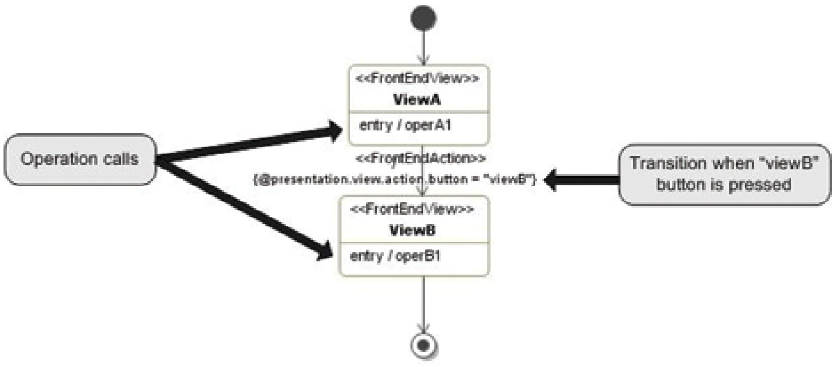


Fig. 7. Application modeling example state transition diagram

case with the trigger class `ContentSorting`. This dependency indicates that the SOAP response will be intercepted and the trigger class operation will be called to modify the message. The conditions under which the operation is called (constraints) and the parameters to be used are both indicated in the OCL expression that accompanies the class (*pre* and *context* part of the expression respectively). The last operation selection depends on the `attrD3` value of `ContextD` to select between the operations available in `WebServiceB`.

The context dependencies correspond to plugin and association entries in the *handler.xml*. For the presented case the structure of the configuration file is depicted in Fig. 6.

The application flow consisting of operation calls to these two web services is depicted in Fig. 7. The *Entry Activity* modeling construct of UML state transition diagrams is used to model the service call. The entry contains information on the response and how the response should be rendered on the application view in the form of stereotypes and tag values (e.g. using the `<<FrontEndField>>` stereotype of the presentation profile). The `<<FrontEndAction>` stereotype refers to the view actions that trigger the transition to the next view and hence application state. In the diagram the transition to the second view of the application is performed when the user presses the *viewB* button as expressed by the tag:

@presentation.view.action.button=”viewB”

6 Conclusions

In this paper the research effort towards a methodology for the model-driven development of context-aware web applications built on web services has been presented. The design of the application is based on the introduced UML profiles. The model transformation to the final platform specific code is performed through the tools developed in the presented work. A use case for the code generation exploiting the Apache Axis2 framework for web services and Spring Web Flow for the application controller has been implemented. Current work includes the full specification of the transformation process and the mapping to different technologies, whereas the potential future work is directed towards the inclusion of privacy-aware mechanisms. Privacy guarantees needed to protect sensitive user data and its dissemination is a major issue of context awareness. Indeed, the issue of privacy has been scarcely handled (e.g. [21]). However, the secure provision and processing of user and service related information remains vital [22]. In order to face this issue, a privacy-aware context profile is being developed. The aim is to integrate in the future this profile in the context-aware service development methodology and extend the context architecture, so that context information processing and retrieval is handled in a privacy-aware manner. A further issue being studied is a potential extension of context dependencies by defining semantics for the exact relations between service and context models. This would require the introduction of ontologies that capture the characteristics of adaptation for the case of applications being studied.

The described work contributes to the advance of the research on model-driven development of context-aware services by providing a methodology that handles context information independently and efficiently in all development stages. It faces a very interesting issue of the latest years and aims in boosting this research area. Through the proposed solution the context management remains independent from the main service logic. The development process is straight forward and adequate independence between the context and service modeling is maintained. This constitutes context model replacement a simple process for the developer.

References

1. Dey, K., Abowd, G.D.: Towards a Better Understanding of Context and Context-Awareness, GVU technical report GIT-GVU-99-22, Georgia Institute of Technology, pp. 3–4 (1999)
2. Gu, T., Pung, H.K., Zhang, D.Q.: A Middleware for Building Context-Aware Mobile Services. In: Vehicular Technology Conference, vol. 5, pp. 2656–2660 (2004)
3. Fahy, P., Clarke, S.: CASS - Middleware for Mobile Context-Aware Applications. In: Workshop on Context Awareness, MobiSys 2004, pp. 304–308 (2004)
4. Keays, R., Rakotonirainy, A.: Context-Oriented Programming. In: Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access, San Diego, CA, USA, pp. 9–16 (2003)
5. Tanter, E., Gybels, K., Denker, M., Bergel, A.: Context-Aware Aspects. In: Löwe, W., Südholt, M. (eds.) SC 2006. LNCS, vol. 4089, pp. 227–242. Springer, Heidelberg (2006)
6. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
7. Object Management Group (OMG), MDA Guide Version 1.0.1 (2003), `http://www.omg.org/docs/omg/03-06-01.pdf`
8. Grassi, V., Sindico, A.: Towards Model Driven Design of Service-Based Context-Aware Applications. In: International workshop on Engineering of software services for pervasive environments: in conjunction with the 6th ESEC/FSE joint meeting, 2007, Dubrovnik, Croatia, pp. 69–74 (2007)

9. Ceri, S., Daniel, F., Matera, M.: Model-Driven Development of Context-Aware Web Applications. ACM Transactions of Internet Technology 7(1), article no. 2, 1–32 (2007)
10. Object Management Group (OMG), Unified Modeling Language (OMG UML) Infrarstructure, v.2.1.2 (2007), `http://www.omg.org/docs/formal/07-11-03.pdf`
11. Ayed, D., Berbers, Y.: UML profile for the design of a platform-independent context-aware applications. In: Proceedings of the 1st Workshop on Model Driven Development for Middleware (MODDM 2006), Melbourne, Australia, pp. 1–5 (2006)
12. Grassi, V., Sindico, A.: Towards Model Driven Design of Service-Based Context-Aware Applications. In: Proceedings of the International workshop on Engineering of software services for pervasive environments: in conjunction with the 6th ESEC/FSE joint meeting, Dubrovnik, Croatia, pp. 69–74 (2007)
13. Keidl, M., Kemper, A.: Towards Context-Aware Adaptable Web Services. In: Proceedings of the 13th international World Wide Web conference (WWW 2004), New York, NY, USA, pp. 55–65 (2004)
14. Object Management Group (OMG), Object Constraint Language OMG Available Specification, v. 2.0 (2006), `http://www.omg.org/docs/formal/06-05-01.pdf`
15. Sheng, Q.Z., Benatallah, B.: ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services. In: Proceedings of the International Conference on Mobile Business (ICMB 2005), pp. 206–212. IEEE Computer Society Press, Los Alamitos (2005)
16. Prezerakos, G.N., Tselikas, N.D., Cortese, G.: Model-driven Composition of Context-aware Web Services Using ContextUML and Aspects. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2007), pp. 320–329. IEEE Computer Society Press, Los Alamitos (2007)
17. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework. Addison Wesley Professional, Reading (2003)
18. Object Management Group (OMG), XML Metadata Interchange (XMI), MOF 2.0/XMI Mapping, v.2.1.1 (2007),
`http://www.omg.org/docs/formal/07-12-02.pdf`
19. World Wide Web Consortium (W3C), Simple Object Access Protocol (SOAP) 1.1 (2000), `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`
20. Kapitsaki, G.M., Kateros, D.A., Venieris, I.S.: Architecture for Provision of Context-aware Web Applications based on Web Services. In: Proceedings of the IEEE conference on Personal, Indoor and Mobile Radio Communications (PIMRC 2008), Cannes, France, September 15-18 (2008)
21. Henriksen, K., Wishart, R., McFadden, T., Indulska, J.: Extending context models for privacy in pervasive computing environments. In: Proceedings of the 3rd International Conference on Pervasive Computing and Communication Workshops (PerCom 2005 Workshops), pp. 20–24 (2005)
22. Weitzner, D.J., Ackerman, M., Darrell, T.: Privacy In Context, Human-Computer Interaction A Journal of Theoretical. Empirical, and Methodological Issues of User Science and of System Design 16(2-4), 167–176 (2001)

DSL Tool Development with Transformations and Static Mappings

Elina Kalnina and Audris Kalnins

University of Latvia, IMCS, Raina bulvaris 29, LV-1459 Riga, Latvia
Elina.Kalnina@lumii.lv, Audris.Kalnins@lumii.lv

Abstract. A tool development framework for domain-specific languages combining mapping and transformation based approaches is proposed in this research project. The combination of both approaches permits to use advantages and eliminate disadvantages as far as possible. First results are described including draft architecture for the framework implementing proposed ideas. A sketch of mapping definition facilities is presented. Initial implementation proposals are described as well. A template based graphical generation language Template MOLA for implementation algorithm description is introduced.

1 Introduction

Currently it is very popular to create and use specialized modelling languages for a domain area. Theses languages are called *domain-specific languages* (DSL). They are developed for users specialized in the concrete area. By using domain-specific languages users can operate with familiar terms. There can be graphical or textual domain-specific languages. Only graphical languages will be considered here. Operational semantics of DSL is also out of scope of this research project. A visual domain-specific language basically consists of two parts – the domain part and the presentation (visual) part. Sometimes they are called also the abstract and concrete syntax respectively.

The domain part of the language is defined by means of the *domain metamodel*, where the relevant language concepts and their relationships are formalized. The domain metamodel is used also for the precise definition of language semantics. Standard MOF [1] or similar notations are used for the definition of domain metamodel.

For the presentation part (concrete syntax) definition there is no universally accepted notation. The same metamodelling techniques are used, but with various semantics. Most frequently, instances of classes in the presentation type metamodel are *types* of diagram elements to be used in the diagram. A concrete set of graphical element types for a diagram definition is called the presentation type model (a typical example is the graphical definition model in GMF [2]).

Tool development for graphical domain-specific languages is time consuming and expensive task. Due to the growing popularity of domain specific languages various graphical tool building frameworks have been developed to improve the tool (editor) building process. Two different approaches are used in these environments. The first option is to use a mapping-based approach. During the tool design this mapping assigns a fixed presentation type model element (a node type, edge type or label type) to a domain metamodel element, by means of which the latter one must be visualized. This solution is quite appropriate for simple cases, where no complicated mapping logic is required. In this case tools for simple DSLs can be developed even during a presentation session. However, DSL support frequently requires much more complicated and flexible mapping logic. One of the reasons is that there is no fixed correspondence between the domain metamodel and presentation types. In this case the second approach is used: to define the correspondence by *model transformation languages*. Transformations define the synchronisation between the domain and presentation models and the tool behaviour in general.

Mapping based frameworks are MetaEdit [3], GMF framework [2], Microsoft DSL Tools [4], Generic Modeling Tool [5] and some other. A pure transformation based framework is METAclipse framework [6]. The other transformation based frameworks Tiger GMF project [7], ViatraDSM framework [8] and GrTP [9] provide also some elements of the mapping based approach.

There exist mapping based and transformation based tools, but usually some parts of the same DSL are suitable for mappings and some for transformations. It means none of solutions is optimal. Problem is that there is no good combined solution. In this paper the combined solution problem is addressed.

The only framework which already proposes some sort of combined solution involving both mapping definition and transformations is ViatraDSM framework [8]. However, a lot of principal issues such as a generic mapping metamodel, seamless integration of static mappings with transformations and user-friendly mapping definition facilities are not solved there. Therefore new ideas for a really satisfactory solution for combined approach to tool building framework are required.

The given paper briefly proposes a new complex solution how to combine transformations and static mappings for tool building. The paper concentrates on architectural solutions and required language facilities. In particular, a new mapping definition language ensuring close mapping and transformation integration is proposed. Some ideas how to implement this solution are also sketched briefly. The implementation is planned as a natural extension of METAclipse framework [6], thus providing advanced facilities for defining presentation type model, mappings and other parts of the tool definition. An interesting facility is introduced for the transformation generation from mappings. It is proposed to use the Template MOLA language to specify the generation algorithm. Template MOLA apparently is the first template based graphical generation language and therefore has a value of its own.

As there is no universally accepted terminology in the area, the second section of this paper begins with a terminology clarification. This section continues with the description of mapping and transformation based approaches as well as related work. In section 3 the original ideas how to combine transformations and mappings are presented. In this chapter the new mapping definition language is introduced. Facilities required to implement this language are described in section 4. Template MOLA is briefly described at the end of section 4.

2 State of the Art in DSL Tool Development

In this section the existing approaches for DSL tool development are briefly described.

2.1 Terminology Explanation

Let us begin with some terminology clarification. Currently different DSL development frameworks use completely inconsistent terminologies, even the terms model and metamodel are used differently depending on the context. For example, the mapping-based GMF [2] speaks only of two layers: model and metamodel, everything the tool builder creates is termed model. This paper combines both the transformations and static mapping context. To avoid misunderstanding, a consistent terminology and its relations to be used in this paper are defined in Fig. 1.

As we can see the domain metamodel is defined using MOF as a meta-metamodel. A domain model is created according to the domain metamodel. It should be noted that alternative domain meta-metamodels used in some approaches in fact play the same role as MOF (and are similar to it).

The situation is not so simple with the presentation part. In every framework there is a fixed presentation type definition environment. Possibilities supported in this environment can be described with a presentation type metamodel. Presentation types for a concrete domain specific language constitute a presentation type model defined according to the presentation type metamodel. Presentation types describe the relevant graphical element types. When data is created in this concrete DSL tool instances of presentation model are created, but data in this model is not an instance in the presentation type model. It is an instance of the presentation metamodel describing supported graphical elements in the tool in general, for example, line, box, label etc. In the presentation type model, for example, we can describe that we want to represent this type as a grey rounded rectangle, with green lines and containing one label. In this case instances of the rounded rectangle, label and colours will be created in the presentation model, with appropriate properties set (according to the presentation metamodel). After instances have been created the user can change the rounded rectangle colour (if this feature is supported by the tool). In this case the presentation

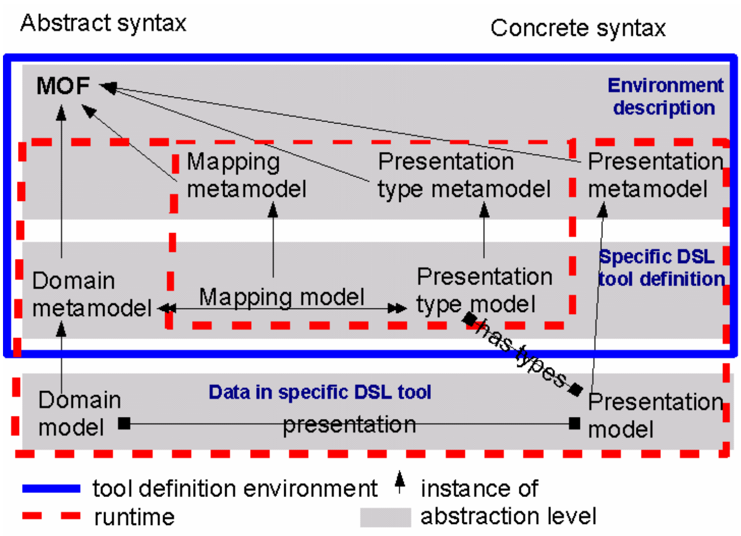


Fig. 1. Terminology definition

model is modified, but it does not affect the presentation type model. The presentation type describes only the default look of this node. That is why presentation model and presentation type model are two separate models.

One more important thing is to define a mapping model. This should be done according to the mapping metamodel. The mapping model describes the relationship between the domain metamodel and presentation types. At the data level mappings are not used directly.

When defining a new DSL tool in a tool definition framework, the user has to define a domain metamodel, a presentation type model and a mapping model. It should be noted that the presentation metamodel is needed directly only if mappings are defined using model transformations. Models required at runtime for the tool created from the definition depend on whether the tool definition framework is an interpreter or generator. If the framework is an interpreter the mapping and presentation type models are needed to interpret them in runtime. If the framework is a generator these models are not needed in runtime because the tool code is generated according to data in these models.

Most of the known DSL tool definition frameworks can be correctly categorized in the framework of this terminology schema.

2.2 Mapping-Based Approach

A *mapping*-based approach prescribes by means of which presentation type model element each domain metamodel element must be visualized. Thus, the graphical tool functionality is basically defined by this mapping. The mapping itself can be defined as a mapping model according to the mapping metamodel. The mapping typically may be complemented by use of constraints, but only at few selected points.

Most of the frameworks (GMF, MS DSL…) use the *generation step*, by means of which language classes are generated in the corresponding OOPL (Java, C#,…) from the involved models. The generated code ensures the relevant synchronization between the domain and presentation models in runtime. If the generated functionality is insufficient, the language code can be extended manually. Actually, mapping may be used without the generation step too - examples are MetaEdit+ [3] and Generic Modeling Tool [5], which are model interpreters.

It must be noted that the mapping approach is easy to use - if the generated code is sufficient (or should be accompanied by a small amount of manual code), the tool definition is mainly declarative and very fast. However, when the presentation type model is dissimilar to domain metamodel, a lot of code in an OOPL must be added. To avoid this, it is a common practice for simple DSLs to create custom domain metamodels nearly isomorphic to the corresponding presentation type metamodels (one class to one node type and so on). However, there can be situations when it is not possible to select the domain metamodel freely, for example, if it is used for compiling, integration with other tools etc.

Mapping definition capabilities of a framework depend on mapping design patterns supported. The most expressive static mapping language is implemented in GMF. But even it is not expressive enough. For example, every domain class mapped to a diagram node must be contained in a domain class mapped to the diagram itself (canvas in GMF). Therefore it is impossible to implement by pure mappings standard UML class diagram where a class is contained in a package (in UML domain) and is visualised in several diagrams independently of its package containment.

Let us take a look at some DSL language examples where mapping approach is clearly insufficient. Evidently, one such group is model transformation languages. A typical example is MOLA [10, 11], which is a graphical language with a lot of semantic dependencies between language elements. It is important to use the native MOLA metamodel as a domain metamodel for the MOLA tool, since only this way complicated syntax checks can be performed during editing and context-sensitive lists of valid references proposed. If the goal of the tool is to create as syntactically correct models as possible, clearly it is impossible to implement this tool using only static mappings. The same can be said about tools for other transformation languages, for example, MOF QVT [12], where the native domain metamodel is even further from the presentation. Another such group could be complicated workflow languages.

2.3 Model Transformation Based Approach

A complete alternative to the mapping-based approach is the *model transformation* based approach. The correspondence between the domain and presentation is defined by *transformations* in a model transformation language, for example, MOLA [10, 11]. These transformations define what modifications must be done in one of the models, if the other one changes (due to user actions or other internal activities). Therefore the correspondence between the domain metamodel and presentation type model may be arbitrarily complicated here. In fact, transformations control the complete tool behaviour.

From the first glance this approach is more complicated to use - though experience shows that programming model element mappings in an adequate model transformation language is much easier than in a standard OOPL. The usability of the approach is ensured also by the fact that a significant part of the transformations are domain-independent and are built only once, as part of the framework itself. Clearly, the transformation driven approach is more time consuming in simple cases.

The first pure transformation based project is the Tiger project [7]. However, a specific domain modelling notation is used there which forces the domain metamodel of a language still to be close to the presentation metamodel. Standard editing actions (create, delete, etc.) are specified by graph transformations which act on the domain model, and the presentation model is updated accordingly. The main goal of Tiger approach is to provide the building of syntactically correct diagrams only.

The most advanced transformation based framework is METAclipse [6] which uses the MOLA transformation language and a powerful presentation engine in Eclipse which is an extension of GEF, GMF runtime and some other plug-ins. It is based on a presentation metamodel specially adapted for defining transformations. The current version of MOLA editor [6] is built on this framework (using a bootstrapping approach). This editor provides an advanced support for ensuring syntactical correctness of MOLA programs and a high usability. The developed editor confirms the suitability of the framework for implementing complicated DSLs.

2.4 Combined Approach

Usually, for some parts of the tool the correspondence from domain to presentation is simple (fit for mappings) and for some complicated (fit for transformations). The best solution would be to combine both approaches. In this case for simple one-to-one relations between domain and presentation the mapping based approach could be used, but for complicated parts model transformations could be written. For example, for the abovementioned MOLA Editor [6] the transformation size could be reduced approximately by 50% if mappings were applicable. Simple visualisation could be defined by mappings, but for complicated consistency maintenance transformations would still be needed.

Currently there are only known two attempts to combine both approaches in a limited way. Frameworks using this combination to a degree are the Tiger GMF Transformation project [13] and the ViatraDSM framework [8].

The Tiger GMF Transformation project [13] (related to the original Tiger project) proposes to extend GMF by complex editing commands. The mapping between domain and presentation models is defined by standard GMF facilities. But new complex model editing commands can be defined by transformations acting only on the domain model. However, this approach does not permit to define more complicated (transformation based) mappings between the domain and presentation, which is the main goal of the approach proposed in this paper.

The ViatraDSM framework [8] is based on the Viatra2 transformation language [14]. In this framework a mapping from domain to GEF-level presentation concepts has to be defined. This static mapping is interpreted by the ViatraDSM engine. The transformation based mapping (defined by Viatra2 rules) can be combined with the static mapping approach. The goal of ViatraDSM seems to be the closest to the proposal in this paper. However, a lot of principal issues are not solved there. First of all, the static mapping mechanisms support only very limited mapping possibilities. Only basic mapping patterns are supported. Mapping and transformation integration possibilities are very limited as well. Each object can be mapped using either transformations or mappings. Mapping definition for ViatraDSM framework has no adequate notation. Solutions to all these issues are the topics of the project described in this paper.

We propose to use a more detailed mapping and transformation integration granularity, for example, to use transformations as preprocessors or postprocessors for mappings. A more expressive mapping language and a mapping definition notation are proposed as well.

There is one more framework GrTP [9] which combines both approaches to a degree, but in a different setting. This framework is based on an advanced presentation type metamodel, by means of which the desired diagram structure is defined. The framework contains a large set of predefined transformations, which implement all standard user actions related to the defined diagram type. All these predefined actions can be extended or replaced by custom transformations. The main application area for this framework is various conceptual modelling languages; therefore there is no built-in support for domain models. If required, synchronisation with the corresponding domain can be supported by custom transformations, but in future a support for typical mappings to domain could be included.

3 Research Project Description

The main topic of the given research project is how to add mappings to a transformation based tool development framework. The METAclipse framework [6] and model transformation language MOLA built by UL IMCS is chosen as the basis for research project realisation. This choice is based on the fact that the framework is completely transformation based, it provides flexible ways of extension and it itself can be used in a bootstrapping manner for implementing the extended features.

To ensure usability of the proposed approach mappings and transformations should be smoothly integrated. The proposed mapping language could be implemented using an interpreter or a generator generating transformations in a model transformation language (MOLA in our case). This implementation decision affects integration possibilities. In both cases extension points where custom transformations can be added to the functionality defined by mappings could be used. If the generator approach is used we can allow also manual modifications of the generated transformations.

The main extension mechanism should be extension points. For this mechanism to be sufficient in most cases, extension points should be chosen appropriately. Extension points should permit to replace or extend the built-in mapping possibilities by custom transformations.

3.1 The Framework from the User Point of View

The proposed tool definition framework will be metamodel based. At the beginning the domain metamodel of a domain specific language should be built (e.g., by MOLA metamodel editor). The next step would be to define the presentation type model and mappings between the domain metamodel and presentation type model. All this will be done using graphical wizard-style dialogs in the tool development framework.

If built-in mapping possibilities are not suitable for some task, the tool builder will be able to select/create custom MOLA procedure (using the built-in MOLA editor). Appropriate parameters to and from this procedure should be passed, to ensure integrity with the mappings. For each extension point parameters passed to procedures used in this extension point are predefined.

When the tool development is complete, the tool builder can press the button "Build tool". Thus the tool executable in one step is obtained. Alternatively, if there is such a need the generated transformations can be edited and then compiled.

3.2 Mapping Definition

Mappings are based on typical mapping patterns. A large set of mapping patterns has been identified in Generic Modeling Tool [5] and they will be reused in this project.

Mapping definition is based on the mapping and presentation type metamodels as the abstract syntax of the "mapping language". The visible form of this language will show up as wizard-style dialogs, which will build instances of these metamodels. Appropriate tool support can be built with a small effort using the METAclipse framework. A more detailed description is given in the next section.

Presentation definition in a graphical tool consists of several parts: property dialogs, diagrams as well as model tree, menus etc. Informal mapping examples mentioned so far all have been related to mapping the domain to diagram element types. Now we switch to another part of the presentation – the property dialogs. It is because the proposed ideas can be easier demonstrated on this part and the corresponding metamodels are smaller. In this paper only an essential subset from the property dialog part of the presentation type and mapping metamodels is briefly sketched (in Fig. 2). We assume here that typical Eclipse-style dialogs are used.

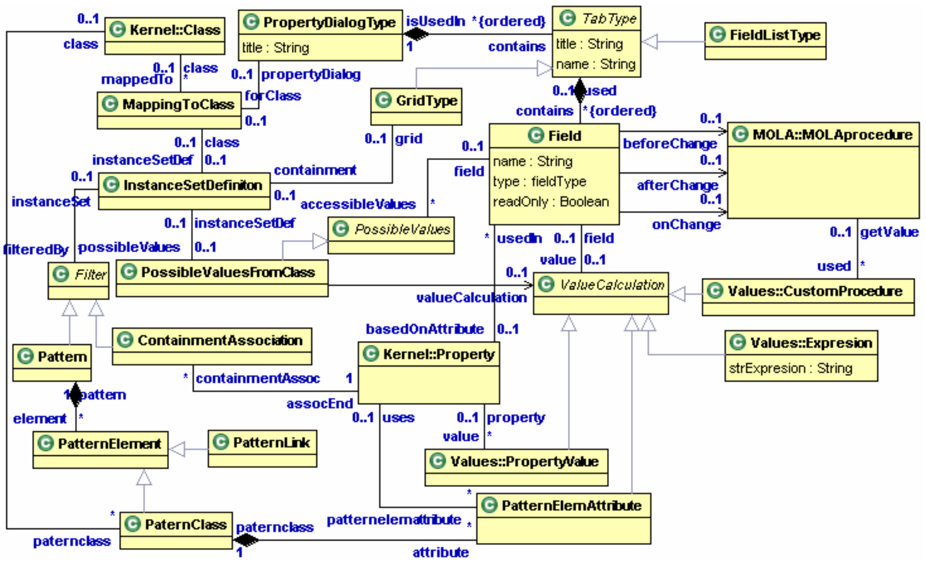


Fig. 2. Mapping and presentation type metamodel subset describing property dialogs

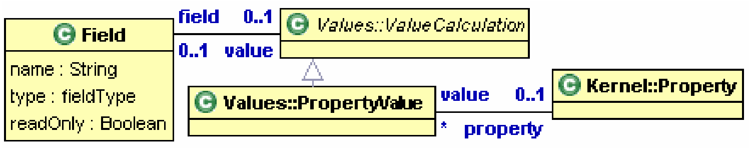
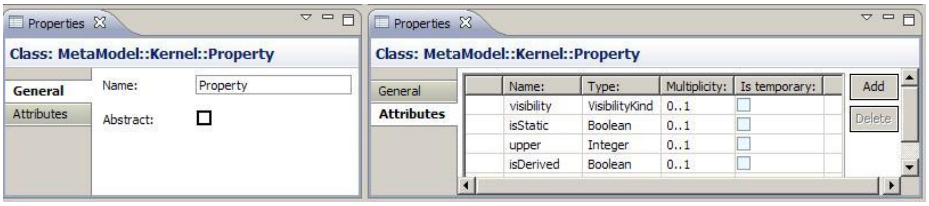


Fig. 3. Metamodel fragment describing design pattern field is based directly on property

When a property dialog for a domain class is to be defined, at first an appropriate property dialog type (i.e., its structure, element types and functionality) is designed, then it is mapped to domain metamodel elements. A property dialog consists of tabs, which can be either a field list (for displaying class attributes and linked class instances) or a grid (for displaying child instance properties in a tabular form). The basic element of both is a field, whose type definition is the central point in the approach. For each field type it must be defined what must be shown there when the corresponding class instance is selected. For many field kinds (e.g. combobox) the valid value set (e.g., a set of appropriate class instances) must be obtained and visualized. Finally, it must be defined what has to be done when the value is modified (in Eclipse-style dialogs the model update follows immediately).

As the metamodel in Fig. 2 shows, for all these situations possible typical cases are defined via mappings to domain metamodel elements (e.g., which class attribute must be visualized in a field in the simplest case, see the fragment in Fig. 3).

The metamodel contains also structuring elements defining various typical ways how these elementary mappings can be combined, e.g., expressions built over elementary mapped values. In all cases the corresponding mapping-based definition can be replaced by a call to a specified custom MOLA procedure. One more novel idea is to


Properties

Class: MetaModel::Kernel::Property

General | Attributes

Name: Property

Abstract: ☐

Properties

Class: MetaModel::Kernel::Property

General | Attributes

Name:	Type:	Multiplicity:	Is temporary:
visibility	VisibilityKind	0..1	☐
isStatic	Boolean	0..1	☐
upper	Integer	0..1	☐
isDerived	Boolean	0..1	☐

Fig. 4. Class dialog example, general and attribute tab

use MOLA patterns for defining custom instance set filters, e.g., for selection of relevant child instances.

For example, we can use this mapping language to describe a property editor for UML 2 class diagrams (based on the standard UML 2 metamodel [1]). For UML *Class* a property dialog type could be defined, consisting of two tabs. The first tab will contain a field list describing the UML Class itself. The attributes *name* and *isAbstract* are directly mapped to fields in this tab. For the attribute *name*, a uniqueness check (within a package) before the change is needed, for this task a custom MOLA procedure can be invoked. The second tab could be a grid describing class attributes (see Fig. 4). In this case, the grid *InstanceSetDefiniton* feature is mapped to the *Property* class. The basic instance selection is via *ownedAttribute* master-detail association and additional filtering is defined using MOLA pattern selecting only those properties which are attributes (but not association ends).

The metamodel part for the diagram mapping and presentation types can be built the same way, only more classes would be present since it is more complicated.

3.3 Mapping and Transformation Integration

For the mapping metamodel the most important task is a seamless integration of mappings with custom MOLA procedures. MOLA is a procedural transformation language, therefore MOLA procedures are chosen as the integration unit. It does not restrict the integration possibilities, since any set of statements can be included in a procedure. Actually it even allows reusing the same procedure in different contexts.

The mapping metamodel granularity and structure should be chosen so that each action could be extended or replaced by an appropriate custom MOLA procedure. The transformation based approach permits to use a more detailed mapping granularity than in traditional mapping based tools.

For each extension point the set of required parameters for custom procedure is predefined. This predefined set should be compatible to the parameter set of the selected procedure.

In Fig. 5 an integration example is given. When a property dialog field is modified a custom transformation can be executed as a preprocessor, postprocessor or instead of the action implied by the static mapping. A custom procedure can be used as well to calculate the field value to be displayed.

This close integration of mappings and transformation based approach is a key factor in reaching the goal when the transformations generated from mapping only need to be combined with the specified custom MOLA procedures, but require no direct manual modification.

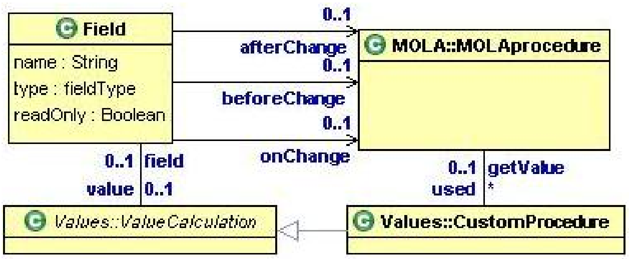


Fig. 5. Metamodel fragment describing mapping and transformation integration

4 Facilities Required to Implement the Approach

To implement the ideas described in the previous section several facilities are required. First of all, a user interface for the mapping definition language should be defined. Then the language implementation is needed. It means an interpreter or code generator for this language is required. This interpreter or generated code should be compatible with the METAclipse framework.

4.1 Mapping Definition Language User Interface

We propose to use wizard style dialogs for the definition of presentation type model and mappings. These wizards will create instances according to the relevant metamodel. The presentation type and mapping definition will be integrated.

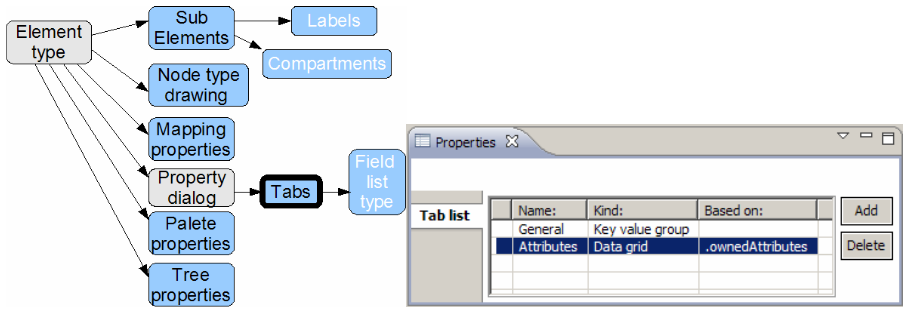
To generate presentation types and mapping for a domain class, the user will be asked to select the appropriate tool design pattern and enter additional properties of presentation types to be created (for property dialog, diagram node type etc.). The relevant mapping instances will be created automatically. The palette element if needed will be created simultaneously as well.

Wizards will be organised in several levels, for the whole domain metamodel (as in GMF) or on one domain class to see or modify the features related only to this class.

In addition to presentation and mapping definition, wizards will allow for complicated cases to select custom MOLA procedures for the relevant extension points. These procedures will be created using the built-in MOLA editor.

A natural way to implement the proposed mapping definition editor in METAclipse framework is to build it as an extension of the existing MOLA tool [6]. Then slightly extended metamodel definition editor can be reused for domain metamodel creation and MOLA editor can be used directly for creating custom procedures.

The mapping/presentation wizard itself could be implemented in several ways. A classical wizard style dialog sequence could be built, but this requires certain extensions to METAclipse property engine. A more interesting and user friendly way could be to create wizard diagrams. The dashboard in GMF [2] could serve as a simple prototype for such diagrams. The possibilities of METAclipse permit to create dynamic wizard diagrams where each node represents some wizard dialog "page". The dialog in such a page can be defined using standard METAclipse property dialog facilities. The edges in such a diagram represent the order in which these pages must



Name:	Kind:	Based on:
General	Key value group	
Attributes	Data grid	.ownedAttributes

Fig. 6. Wizard diagram example, for domain class mapped to Node

be visited. Next nodes and edges will be created and existing ones enabled/disabled in response to the values the user has entered in the current node. A simplified sketch of a wizard diagram for a domain class mapped to node can be seen in Fig.6. It is assumed that the user currently defines tabs for the property dialog.

The same visual representation can be used to modify the defined mappings. After opening the appropriate wizard diagram the user can select a node and update properties. If this modification influences dependencies to other wizard nodes the user is asked to update these nodes as well.

We can think about other mapping visualisation possibilities too. For example, a "mapping diagram" similar to the one in Microsoft DSL Tools [4] can be used, with domain metamodel in one side of the diagram and presentation type model in another, and with mapping lines connecting them. A lot of improvements are possible for this idea. The domain part could be visualised by a standard class diagram. A palette element (if needed) can be shown together with the presentation type. A presentation type can be visualized close to the node with this type. Instead of a label a short form of the template how this label value will be calculated can be shown. Subelement mappings could be shown in a similar way too.

4.2 Mapping Language Implementation

As it was already mentioned a mapping language implementation is needed and this implementation can be done via an interpreter or generator.

One of the solutions is to create a generator generating MOLA transformations from the defined mappings. MOLA is selected as the target language since it is the base language of the METAclipse framework and custom transformations will also be in it. In addition, it will be easier for the user to modify the generated transformations if needed.

The most straightforward approach would be to define this generator in MOLA language as well. However, a more interesting solution requiring less effort to be implemented can be provided in this project. It is possible to define a "MOLA template language". It will be a template language, combining executable parts in MOLA with templates for transformations to be generated. The first experiments (about 10 % of generator written) show that generator algorithms in this Template MOLA could be defined quite easily. It should be noted, that the Template MOLA has a value of its own as a general purpose macro-processor for MOLA. A more detailed description of this language is given in the next section. One more aspect is that Template MOLA would permit to modify the generator procedures themselves more easily, for example, to adapt the framework to some specific kinds of DSLs.

Another possible way to implement the transition from mapping definitions to MOLA would be to build a universal interpreter in MOLA which would directly interpret them. Some experiments show that the interpreter would consist of procedures quite similar in form to those used in generator. Certainly, some true extensions to MOLA language and compiler would be required in this case. Also, there would be impossibile for tool builder to modify the "generated" code. However, the total effort for interpreter approach could be less.

4.3 Template MOLA

The "MOLA template language" is a direct generalisation of popular textual template languages (of the kind model-to-text) to graphical languages. The planned Template MOLA language would contain two kinds of MOLA statements: standard ones to be executed during the generation process and those to be "copied" to the generated "code" (in fact, model) with template expressions replaced by the appropriate generation time values. Some interesting solutions could appear here, for example, how to generate a set of similar procedures from one template procedure. The template part of the language requires some natural extensions of MOLA syntax, for example, reference to a parameterized class in the MOLA pattern definition.

The metamodel for transformations in Template MOLA consists of three parts. The first one is the generation time part. As already mentioned, there are statements executed in generation time in Template MOLA. These statements are similar to traditional MOLA and elements used in them should reference the generation time part of the metamodel. In the context of tool building the mapping and presentation type metamodels should be used as this part. Then there is the template part of Template MOLA. Situation with this part is more complicated. There are constant and variable parts of template statements. By constant part we understand elements to be copied directly to the generated MOLA code. Accordingly, there must be also the constant template part of the metamodel. The constant part of template statements should reference only the corresponding part of the metamodel. This metamodel part must be present in the Template MOLA definition environment and must be copied to the runtime metamodel. In the given context, the presentation metamodel is this constant part. There is also the variable part of template MOLA (elements with parameterized types – in angle brackets). For example, the parameterized type "<class>" means that some class should be used here. The generator part during its execution has to substitute this parameter by a reference to an appropriate class. The variable part of metamodel must be provided before the generator is run. In the given context this variable part of metamodel is the domain metamodel (which is supplied by tool builder before the generator execution).

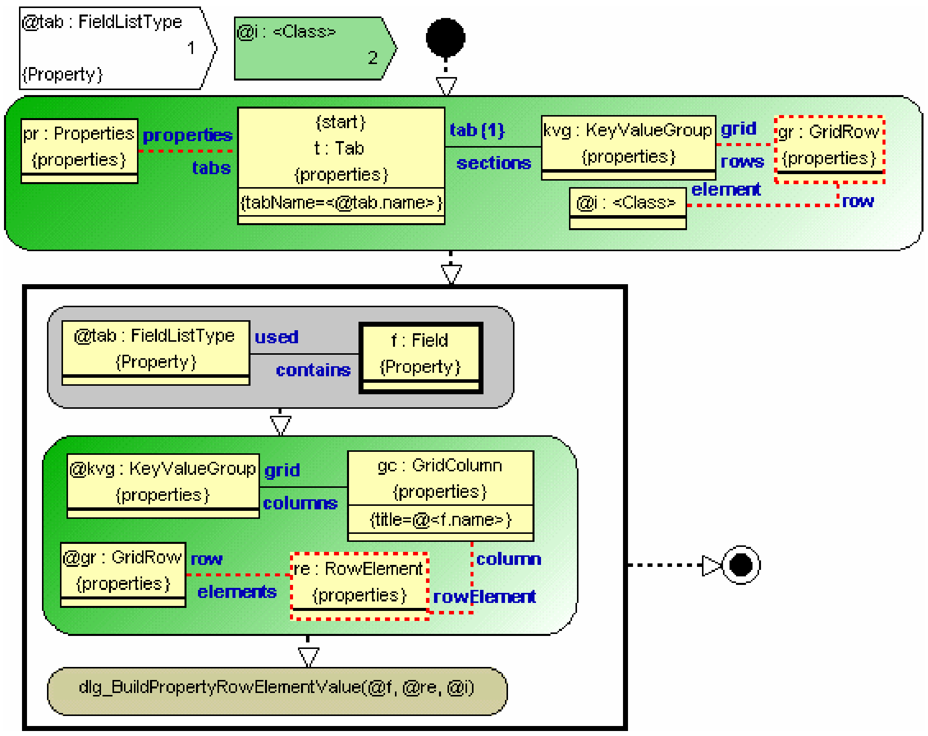


Fig. 7. Template MOLA example

You can see a Template MOLA procedure example in Fig. 7. The procedure generated from this template procedure creates a property dialog tab which actually is of *FieldListType*. We assume that the basic structure of property dialogs is already created during initialization. This procedure is called when some property dialog should be visualised, its goal is to fill the “prefabricated” dialog structure with data. The first task is to find the already created tab by its name. Then for each field its value is set using the procedure *dlg_BuildPropertyRowElementValue*.

This procedure has one generation parameter *@tab* and one template parameter *@i*, the template parameter has a parameterized type. Another Template MOLA procedure invoking the given one will supply a particular class for this template parameter (in fact, a domain class), therefore in the generated procedure this class will be used as the type of this parameter. The procedure begins with a template rule. Four class elements have types from the presentation metamodel (the fixed part of template metamodel) and one is a reference to the template parameter whose type is substituted the same way as in the parameter itself. During generation this rule is copied to the generated MOLA code, replacing the type of the parameterized element in the described way. Then the procedure contains a loop executable in generation time. It is executed for each *Field* in the *FiledListType*. This loop contains a template rule and a procedure call. In this template rule all elements have types from the fixed part of template metamodel. For each loop iteration one copy of this template rule is created.

In the generated code only *@f.name* template expression is replaced with its generation time value. The procedure call means both the generation time call and a generated call, the generated call (to the corresponding generated procedure) will contain only the template parameters. Control flows are generated as well in an appropriate way.

The implementation of template MOLA itself would also not be very complicated. The Template MOLA editor could be built in METAclipse framework using the MOLA editor as basis. The experience in creating MOLA editor shows that it could be done quite easy and would take about one men-month. Also a compiler for Template MOLA is needed. It could be implemented in two steps. The first step would be a "preprocessor" converting template MOLA to traditional MOLA. Certainly, only the abstract syntax form (model) of MOLA can be easily generated, but this is sufficient for the subsequent compilation. In the second step the existing MOLA compiler could be used. The preprocessor converting template MOLA to ordinary MOLA seems to be not very complicated. We have written approximately 10 percent of this preprocessor and these experiments are very promising.

5 Conclusions

The overall goal of this research project is to develop the scientific basis required to create a DSL tool development framework with integrated mapping and transformations support. The main target is to develop language and metamodel facilities for this framework. Only an experimental version of the framework is planed to validate the proposed approach and languages.

Currently draft requirements for such a tool development framework have been developed. The first version of mapping definition and generation/interpretation languages has been developed. These languages should be improved and tested on real life examples. Detailed architecture of the framework should be developed including tool support for the proposed languages.

Actually, ideas and languages described in this paper could be used in terms of other frameworks as well. It would be easier to apply these ideas to transformation based frameworks because their structure is quite similar to METAclipse.

Tool development framework implementing ideas described in this paper could be built with a reasonable effort. It would permit to reduce the transformation size for MOLA Editor approximately by a half. Tool for MOF QVT Relational graphical form [12] (using the original metamodel from OMG) could also be built relatively easy using this framework. The framework would be suitable for building advanced workflow editors as well, for example advanced BPMN implementation with syntax directed editing.

Though there are many open questions, first experiments (redefining some parts of MOLA tool) seem to be very promising.

Acknowledgments. The authors would like to thank Agris Šostaks and Edgars Celms for their help and suggestions.

References

1. Meta-Object Facility (MOF), `http://www.omg.org/mof/`
2. Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language Toolkit, Rough Cuts. Addison-Wesley, Reading (2008)
3. MetaEdit+, `http://www.metacase.com/`
4. Cook, S., Jones, G., Kent, S., Wills, A.C.: Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley, Reading (2007)
5. Celms, E., Kalnins, A., Lace, L.: Diagram definition facilities based on metamodel mappings. In: OOPSLA 2003, Workshop on DSM, Anaheim, California, USA, pp. 23–32 (October 2003)
6. Kalnins, A., Vilitis, O., Celms, E., Kalnina, E., Sostaks, A., Barzdins, J.: Building Tools by Model Transformations in Eclipse. In: Proceedings of DSM 2007 workshop of OOPSLA 2007, Montreal, Canada, Jyvaskyla University Printing House, pp. 194–207 (2007)
7. Ermel, C., Ehrig, K., Taentzer, G., Weiss, E.: Object Oriented and Rule-based Design of Visual Languages using Tiger. In: Proceedings of GraBaTs 2006, p. 12 (2006)
8. Rath, I., Varro, D.: Challenges for advanced domain-specific modeling frameworks. In: Proc. of Workshop on Domain-Specific Program Development (DSPD), ECOOP 2006, France (2006)
9. Barzdins, J., Zarins, A., Cerans, K., et al.: GrTP: Transformation Based Graphical Tool Building Platform. In: Proc. of the MDAUI Workshop of MoDELS 2007, Nashville, Tennessee, USA, October 1, 2007. CEUR Workshop Proceedings 297 CEUR-WS.org (2007)
10. Kalnins, A., Barzdins, J., Celms, E.: Model transformation language MOLA. In: Aßmann, U., Aksit, M., Rensink, A. (eds.) MDAFA 2003/2004. LNCS, vol. 3599, pp. 62–76. Springer, Heidelberg (2005)
11. UL IMCS, MOLA pages, `http://mola.mii.lu.lv/`
12. MOF QVT Final Adopted Specification, OMG, document ptc/08-04-03 (2008)
13. Taentzer, G., Crema, A., Schmutzler, R., Ermel, C.: Generating Domain-Specific Model Editors with Complex Editing Commands. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) AGTIVE 2007. LNCS, vol. 5088, pp. 98–103. Springer, Heidelberg (2008)
14. Visual Automated Model Transformations (VIATRA2), GMT subproject, Budapest University of Technology and Economics, `http://dev.eclipse.org/viewcvs/indextech.cgi/gmthome/subprojects/VIATRA2/index.html`

Current Issues in Teaching Software Modeling: Educators Symposium at MODELS 2008

Michał Śmiałek

Warsaw University of Technology,
Warsaw, Poland
smialek@iem.pw.edu.pl

Abstract. Software modeling is an increasingly popular method among software development teams to overcome the problems associated with complexity of contemporary software systems. It can be argued that teaching modeling to software engineers is as important as teaching maths and physics to general engineers. This lead many academic and industrial centers to introducing modeling courses to their curricula. Unfortunately, it seems that education does not yet support the modeling paradigm well enough, thus limiting its acceptance as a mature method of developing software systems. This symposium sought for innovations on teaching modeling that would show its benefits in attractive and pedagogically effective ways. Three areas that need improvement were identified: placing modeling in the software engineering curriculum, teaching model semantics, and tool support for teaching modeling. Several pedagogical solutions in these three areas were presented. The symposium concluded with several recommendations on such improvements, with the most important being: "let the students play with models".

1 Introduction

Modeling is an important part of any software development effort, even if the developers do not realize that they are in fact doing modeling by eg. scribbling quick diagrams on whiteboards. It allows developers to abstract over complex problems and create "visual maps" of the "territory" covered by the resulting programs. However, usually, models are used only informally to support programming with additional (often hand-written) "visual notes". It can be noted that most of the engineering disciplines have a common language that allows engineers from various backgrounds communicate effectively. It can be argued that software engineering also needs such a "lingua franca" and modeling languages serve this purpose very well. Unfortunately, in order for modeling to become to software engineering what maths is for general engineering, much wider knowledge of modeling semantics is needed.

Although it is fairly widespread, the approach of using models in a systematic way which includes automatic model transformation and code generation has not yet got through to industry to the degree which would satisfy the advocates of the approach. This might be caused by lack of high quality, pedagogically effective

MDE courses. Assuming we are convinced ourselves, we need to convince the students – prospective software engineers – that MDE gives significant benefits over other approaches. The students need to experience these benefits and gain appreciation leading to usage of MDE in their day-to-day business. Convincing the students of MDE is not easy, as it is usually treated as a "nice to have" feature within a software engineering project. We should thus discuss ways to convince people within the project teams to treat the model-driven approach as a "must have" that introduces additional, necessary level of abstraction. This in turn leverages understanding of complex systems also through the existence of the above mentioned "lingua franca". We need to convince students who normally use eg. only Java IDEs that they also need modelling plug-ins which would not only give a visual map of their Java programs but also allow for better organization of the path from user requirements to code.

2 Organization, Selection and Thematic Areas

The symposium was organized around three types of submissions: full papers, short position papers and informal teaching technique descriptions. Formal submissions resulted in classical paper presentations and associated discussion. Informal submissions resulted in "tricks and tips" sessions where several educational artifacts were presented and discussed. From the submitted formal papers only 8 could be selected, and a thorough review process was needed. The selected paper submissions formed three thematic groups: teaching model semantics, tool support for teaching models and modeling course concepts.

Within the first area, there were presented approaches to show to the students that modeling (specifically in UML) is more than just drawing nice (meaning also: syntactically correct) diagrams. Within the proposed courses, the role of modeling language semantics is presented to the students. This is important, as only presenting model semantics in a pedagogically effective way can supply the students with arguments for using models in practice. An interesting approach to tackle the above issue is presented in one of the papers selected for this volume - "The UML is more than Boxes and Lines". Throughout the course presented in this paper, the students get to know consistency rules for UML specifications in different software development contexts. The presented approach is unique through the use of inconsistencies introduced deliberately by the teacher to the handed-out models. In another paper presented during the Symposium an attempt to simplify UML's meta-model for teaching purposes was presented. The third paper in this area presents an approach to teach semantics of model transformations through showing small variations in models.[1]

The second group of papers stresses the role of tools that support teaching. This does not only pertain to the obvious role of CASE tools. Novel approaches

[1] All the papers are published in a volume issued by the Warsaw University of Technology; abstracts are available at the Symposium website: http://www.iem.pw.edu.pl/edusymp08/.

use tools to give certain pedagogical feedback on the models created by students. While creating their models within CASE tools, students receive instant comments that help in improving their modeling style. Such tools relieve the teachers and help in organizing courses with many attendants. They can also help in preparing fair and transparent criteria for assessing models created by many students. Such an approach is presented in the second paper selected for this volume - "Automatic Checklist Generation for the Assessment of UML Models". This paper presents a tool that automatically generates a checklist for modeling assignments based on natural language description. With the generated checklist, grading the models prepared by the students is significantly facilitated.

The final group of presentations treated new course ideas described through short papers. This includes ideas on how to cope with large "volumes" of students (up to 1000 per course) and approaches which bring the modeling theory closer to practice by introducing project assignments that emulate real life in a software development organization. These course ideas were supplemented by the "tricks and tips" which included ideas on: how to present to the students the syntax and semantics of interaction diagrams in a way that actively involves the students; how to use the official chess specification as the basis for a modeling assignment; various assignments associated with modeling.

3 Symposium Results

The symposium concluded with discussion in groups formed around the above three thematic areas. The discussion concentrated on finding more effective ways to promote the MDE paradigm through software modeling education. It resulted with several recommendations for the prospective curricula and courses on software modeling:

– Introductory modeling should be taught as early as possible within the software engineering curriculum (as compared to teaching maths or physics in general engineering curricula). It could be taught even before or just after the basic programming course.
– Modeling courses should use good, convincing examples. Toy examples often found in textbooks are not satisfactory for attracting the students. The examples should start with natural language descriptions from which the models could be discovered.
– Modeling should be taught in the context of software engineering process. Both agile and formal methodologies should be covered for modelling. Model Driven Development should be taught as a compulsory course at the master level.
– It is very important to teach properly the semantics of modeling languages (specifically UML). This semantics should be presented for different contexts that the modeling language is used in (business, requirements, design). UML has to be taught with care due to ambiguous semantics caused by universality of the language.

- The students learning modeling should be able to "play" with the modeling languages similarly to "playing" with the programming languages. In order to get convinced, the students need to see concrete (eg. executable) results of their modelling in a specific assignment.
- More than just a model drawing tool is needed to practice real modeling. In order to see the real benefits of modeling and model driven development, the students need to see model transformations and executable models through appropriate tool usage.
- Modeling should be taught also to non-software developers. Communication through visual models is a skill necessary for many people in contemporary business and research environments.

4 Conclusion

The most important message for modeling educators that comes out of the Educators Symposium at MODELS'08 can be summarised in a short motto: "make models fun". Only by allowing the students to have fun with the models we can convince them to use modeling techniques in their day-to-day activities. It is also important to start modeling as an activity that abstracts over various programming languages and software technologies. Only this way, the models can fill the gap that separates natural language descriptions of a problem with the problem solution in a specific programming language. This leads to the ideal educational scenario, where the students can first play with the initial, natural-language-based models, then transform them to more detailed ones and finally execute them just like for programming languages. This allows to achieve "concrete" results that give personal satisfaction to the students, and makes the learning process "fun". Hopefully, the above scenario can be implemented in various educational contexts thus leveraging the spread of modeling approaches in the software development industry.

Acknowledgement

I would like to thank all the members of the Program Committee for their help in selecting the most relevant papers which highly contributed to the symposium's success. I would also like to thank the Symposium participants for their involvement in preparing excellent formal and informal presentations and for interesting discussions which lead to this summary text.

The UML Is More Than Boxes and Lines

Yvan Labiche

Carleton University, Department of Systems and Computer Engineering,
Software Quality Engineering Laboratory
1125 Colonel By Drive, Ottawa, ON K1S5B6, Canada
labiche@sce.carleton.ca

Abstract. The Unified Modeling Language (UML) is now the de-facto standard for the analysis and design of object-oriented software systems. There is a general consensus among researchers and practitioners that the UML could have a stronger semantic content. However, even the semantics of the UML, as described for example as well-formedness rules in the UML standard documentation, is not very well-known to many practitioners. As a result, practitioners often perceive the UML merely as a graphic tool. This paper discusses the apprenticeship of the UML semantics and presents a pedagogical method to help students overcome their limited view of the UML language as merely a set of annotated boxes and lines and to allow them to discover UML semantics.

Keywords: UML, well-formedness rules, consistency, teaching, laboratory.

1 Introduction

The Unified Modeling Language (UML) [11] is now the de-facto standard for the analysis and design of object-oriented software systems [12]. Every researcher working (extensively) on, and possessing an intimate knowledge of, the UML would agree that UML is much more than a set of annotated boxes and lines. Indeed, these annotated boxes and lines come together to reflect the meaning of a software, that is, the UML has semantics. The semantics of the UML is expressed through its metamodel and in particular through so-called well-formedness rules, which describe in plain language and often using the Object Constraint Language (OCL) [10] the constraints that UML model elements have to satisfy. For example, a well-formedness rule states that generalization hierarchies shall be directed and acyclical [11]. This well-formedness rule only involves model elements that belong to one UML diagram, specifically the class diagram, whereas other well-formedness rules involve different UML diagrams. For example, the signature of a message in an interaction diagram must either refer to an operation (i.e., an operation of a class in a class diagram) or a signal (i.e., a signal class in a class diagram) [11]. Therefore, although some researchers and practitioners would like to improve the UML semantics—and there have been many attempts at doing so (e.g., [8])—claims that the UML is devoid of semantic content do not hold true. It results that even the most rigorous application of UML syntax cannot ensure logical outcomes. The correct practice of the UML necessarily entails the simultaneous understanding of UML syntax and semantics.

Despite UML's potential for carrying meanings, current practice (in schools and in the profession) often shows that the semantics of the UML, even the one specified in the UML metamodel, is not (very well) known to practitioners. Many practitioners and students are simply not aware of the semantics of the UML and therefore perceive the UML merely as a graphic tool that allows them to freely draw annotated boxes and lines. There are probably two main reasons for this reduction. On the one hand, UML semantics is not necessarily taught to students during their degree (and few textbooks explicitly mention the problem of well-formedness of UML diagrams), and on the other hand, tools do not necessarily enforce well-formedness rules (even the ones of the standard), and as such mask the need for a semantic understanding.

In this article, we report on an attempt we have made to address the first possible root cause of this erroneous perception. We created laboratory material to heighten student awareness of the problem of the well-formedness of UML diagrams. The laboratory material consists of a UML analysis document (with use case, class, interaction, state machine diagrams and a data dictionary) in which well-formedness problems, introduced by the instructor, were to be discovered by undergraduate students. In this article we describe this laboratory material and report on our experience of its use since 2003. In particular, though the laboratory material could be improved, results showed that we did achieve our objective: to heighten student awareness of the problem of the well-formedness of UML diagrams. Students were indeed able to identify a number of well-formedness problems by themselves. This is particularly important in the context of Model Driven Engineering [7] where models are the primary artifact of interest.

The rest of the paper is structured as follows. Section 2 puts this paper into context by describing the course in which the laboratory exercise took place. Section 3 describes in detail the laboratory material, and observations are discussed in Section 4. Conclusions are drawn in Section 5.

2 Course Context

The *Software Engineering* course in which the laboratory sessions took place is a 4th year course offered to Computer System Engineering and Software Engineering bachelor students. It is composed of three hours of lecture a week and three hours of laboratory every other week, over a period of eleven weeks. Its objective is to go through the different phases of software development, from requirement elicitation to design, code and testing with the support of the UML notation [3]. This course is preceded in the curriculum by the *Systems Analysis and Design* course, which is mostly dedicated to teaching the UML notation, and it is a pre-requisite to the *Analysis and Design Laboratory* course which is a laboratory-based course where students put what they have learnt into practice. (See calendar descriptions of the three courses in Table 1).

As such, at the beginning of the Software Engineering course, students possess a relatively solid command of the UML notation (use case, class, interaction, activity, and state machine diagrams mainly), as well as the OCL, but have had near to no exposure to complete analysis and design documents, and have not seen how the UML notation can be used during the many software development phases.

Table 1. Course calendar descriptions

Systems Analysis and Design	Creating requirements specifications prior to designing and implementing complex software systems. Software development lifecycles, role of requirements analysis; functional decomposition, dataflow modeling; database modeling, entity-relationship diagrams; finite state machines; object-oriented analysis; use cases, use case maps; project management; introduction to software design.
Software Engineering	Review of software lifecycles and requirements analysis. Software design, with emphasis on methods for real-time systems. Testing, verification and validation, quality assurance and control. Project planning and management. Maintenance and configuration management. Software reuse during design and maintenance.
Analysis and Design Laboratory	Applying the full spectrum of engineering and programming knowledge acquired in the program through team projects in the laboratory. Practice in doing presentations and reviews. Lectures will discuss software engineering issues as they relate to the projects, from a mature point of view.

Additionally, students embarking on the Software Engineering course have not been introduced to the fact that UML diagrams allow the decomposition of an analysis/design problem into sub-problems focusing on one aspect of the problem at a time, such as structure or behavior. Relatedly, the students have no experience in grasping that these diagrams, which together describe one software system, are to be coherently inter-related and consistent with each other. And from here, new challenges arise: in designing diagrams, students make numerous mistakes of incompatibility or inconsistency. In other words, they are aware of the UML syntax and some well-formedness rules that involve one diagram only and that relate to the correct use of the UML graphical notation (e.g., generalization hierarchies shall be directed and acyclical [11]), but are not necessarily aware of the well-formedness rules that involve several diagrams and which make up a coherent analysis/design document. For instance, our experience shows that at the beginning of the Software Engineering course, students do not systematically ensure that signatures of messages in interaction diagrams refer either to operations (i.e., an operation of a class in a class diagram) or to signals classes (i.e., a signal class in a class diagram).

3 Laboratory Material

A first note: the well-formedness rules discussed in this article are not restricted to the ones specified in the UML standard [11]. They more generally address the issue of the consistency between different parts of an analysis/design document [3]. Therefore, we will use the term *consistency rule* instead of well-formedness rule in the remainder of this article, unless we refer to the rules of the standard.

It was decided that the first laboratory session in the Software Engineering course would be dedicated to heighten student awareness of the problem of the consistency of UML diagrams, and specifically consistency rules that involve different parts of an analysis/design UML document. To do so, the laboratory material included:

1. An introductory (30mins) lecture on consistency rules involving different parts (including UML diagrams) of an analysis/design document (Section 3.1);
2. The construction (by the teaching assistants and the course instructor) of an analysis document for a realistic system (Section 3.2);
3. The insertion of inconsistencies in the analysis document (Section 3.3).

During the three-hour long first laboratory session, the students then:

a. Attended a 30mins lecture on UML diagrams consistency;
b. Were given the task, during the remainder of the laboratory, of finding as many consistency problems as possible in the analysis document with the help of the consistency rules informally described during the lecture. They were not asked to fix the identified problems, as this was later discussed in class.

3.1 Introductory Lecture on UML Diagrams Consistency Rules

Since the UML analysis document provided to the students comprised use case, class, interaction and state machine diagrams as well as use case descriptions and a data dictionary[1] (see Section 3.2), the consistency rules discussed during the introductory lecture focused on these different views only. The objective of the introductory lecture was not to introduce the UML metamodel in detail, describing every single well-formedness rule it contains. Our experience is that introducing the notion of metamodel, going through the UML metamodel and discussing its well-formedness rules would take more than half an hour (the duration of the introductory lecture). Additionally, some well-formedness rules of the standard documentation are already known to the students: e.g., the fact that a generalization hierarchy is acyclical. Last, the UML standard documentation is not meant to be read by bachelor students, and is dry reading anyway. Instead, we decided to give an informal description of some consistency rules, focusing on those that involve more than one part of a UML analysis/design document and that we consider important when designing a software system with the UML notation. In other words the list of rules is not meant to be complete. These consistency rules are summarized in Table 5.

Table 5 is basically a square, symmetrical matrix. The reason for this symmetry is that in order to ensure that a specific diagram (say an interaction diagram) is consistent with another diagram (say a class diagram) a two-way analysis is required. For instance, all the operations (in message signatures) of an interaction diagram should appear in a class of a class diagram, and operations in classes of a class diagram should be used somewhere in interaction diagrams. Note that not satisfying these consistency rules may not indicate a design problem: it may simply be a case of the UML model not being complete (e.g., an interaction diagram is not complete if it does not include all the possible scenarios and therefore some of the operations of classes are not used in interaction diagrams). Failing to satisfy a consistency rule may also be due to the level of detail of diagrams: the designer may decide to omit some messages (and therefore the use of some operations) in interaction diagrams.

Note also that these consistency rule descriptions could be more precise. For instance, when comparing an interaction diagram to a class diagram (i.e., row "Interaction diagram" and column "Class diagram") one could add that an operation used in a message signature should belong to the class which is the type of the target of the message. Though this information does not appear in the table for reasons of manageability of size, it was discussed during the introductory lecture.

[1] Note that such a data dictionary, with descriptions of use cases, classes, operations, attributes, associations, contracts, …, could be automatically produced by UML case tools. It can therefore be considered part of the UML model.

Last, these rules have four main origins. First, some come from the UML standard itself [11], such as the "Interaction diagram" / "Class diagram" example discussed above. Some come from current practice (e.g., based on experience and designer feedback [1]) or are recommended in software engineering textbooks [3]. The fact that classes and operations should be documented in a data dictionary is an example of such tradition or practice-based rules. Third, some come from the literature on the UML notation, like the idea that state machines can be synthesized out of a collection of scenarios (e.g., [2, 15]) thereby suggesting some consistency rules between interaction and state machine diagrams (e.g., row "State machine diagram", column "Interaction diagram" in Table 5). Indeed, an interaction diagram specifies co-operations between objects and those co-operations should trigger (legal) objects' state-based behavior as specified in state machine diagrams. Fourth, some come from our own experience of UML-based design; this is the case for the consistency rule between scenarios in interaction diagrams and operations' pre and post conditions in the data dictionary (i.e., row "Interaction diagram", column "Data dictionary").

To illustrate this last point, consider the example of Fig. 1. Though greatly simplified, Fig. 1 (a) shows a class diagram and one set of contracts for operation `addUser(name:String):Integer`. The predcondition, i.e., what the client of the method has to ensure according to Design by Contract [9], states (in OCL) that the value of parameter `name` does not correspond to an existing user. Figures (b) and (c) show two different interaction diagrams (excerpts). The first one is consistent with the operation contract: the client (not shown in the diagram) ensures that the precondition is met before calling `addUser()` by calling `findUser()` beforehand. However, the second one is not: nothing ensures that the precondition is satisfied and therefore, according to Design by Contract [9] the result of `addUser()` is unexpected. The "Interaction diagram" / "Data dictionary" rule therefore demonstrates to students that specifying class and operations responsibilities under the form of contracts has an impact on the construction of interaction diagrams.

Since it was the first time the students were introduced to a long, almost complete analysis document (see the next section), the introductory lecture additionally gave the students some guidance as to how to read an analysis document. One approach could have been to teach the students an existing inspection technique that has shown to have important benefits [5, 6], tailored to a UML context. Instead, because time

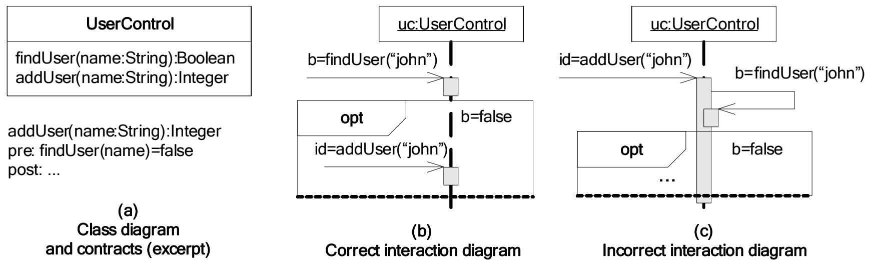


Fig. 1. Illustrating coherence between scenarios and operation contracts

was limited at the beginning of the course and in the first laboratory, the students were encouraged to follow the guidelines below:

1. Do not attempt to read the document sequentially from the first page to the last page.
2. Follow an iterative procedure, to incrementally increase your understanding, as follows:

   i. Take a look at a use case that seems to be one of the most important ones from the use case diagram and use case descriptions.
   ii. Identify the corresponding interaction diagram to learn how objects cooperate to achieve the tasks of the use case.
   iii. Look at the class diagram to understand how the corresponding classes are structurally related.
   iv. Look at the data dictionary (i.e., class descriptions and operation contracts) to precisely understand the object interactions.
   v. Look at state machine diagrams where necessary to discover state-based behavior.
   vi. Repeat those steps for another use case.

3.2 The Analysis Document

The analysis document distributed to the students during the first laboratory session of the Software Engineering course describes the (incomplete) analysis of a Cab Dispatching System. This system can be used to dispatch cabs following clients' requests in real-time, to pre-book cabs in advance and dispatch them when it is time to do so. The analysis document is 40 pages long and contains a use case diagram, use case descriptions (for each use case) following a specific template [3], a plain language description of the seven actors, two class diagrams (one with only the entity classes and one with all the entity, control and boundary classes[2]), interaction diagrams for three of the use cases, state machine diagrams for three of the entity classes, and a data dictionary describing classes (with class invariant specified in OCL), attributes, and operations (with OCL contracts) in plain language. Other detailed information about this analysis document is available in Table 2.

Table 2. Details about the Cab Dispatching System analysis document

Total Number of classes	20	Number of use cases	10
Number of entity classes	6	Number of actors	7
Number of control classes	4	Number of interaction diagrams	3
Number of boundary classes	10	Number of state machine diagrams	3
Number of attributes	25	Descriptions of use cases and actors	7 pages
Number of operations	59	Data Dictionary	23 pages
Number of class relationships	21		

[2] Entity classes represent persistent or long-lived information tracked by the system whereas control classes perform the tasks supported by the system and boundary classes show interactions between the system and actors [3].

The document was left incomplete (e.g., not all the use cases have an interaction diagram) for several reasons. As a complete document would very likely have more than 100 pages, students would feel overwhelmed and spend most of their time trying to understand the whole system instead of investigating consistency. We did not include all the classes, attributes and operations, and we did not include all the interaction and state machine diagrams. However, the analysis document is complete enough: it fully describes (with use cases, use case description and interaction diagrams) the main functionalities of dispatching a cab for an immediate request, booking a cab in advance (e.g., a day before), and dispatching a pre-booked cab. The entity classes involved in the interaction diagrams that have a state-based behavior are described with a state machine, and all the classes, attributes and operations involved in interaction and state machine diagrams appear in the data dictionary.

3.3 The Inconsistencies Introduced in the Analysis Document

The analysis document was constructed by a graduate student well-versed in UML-based object-oriented analysis and design, and reviewed by the instructor and the teaching assistants of the course. During the reviewing process, a number of inconsistencies were identified and fixed. These were the starting point of a larger set of inconsistencies that were eventually re-inserted in the document before distributing it to the undergraduate students. Aside from inconsistencies due to the document's incompletion (these were quickly weeded out through in-class instructions), the inconsistencies the undergraduate students were asked to identify were not arbitrary: they are the real errors that a (junior) designer is likely to make.

A total of 57 inconsistencies were introduced in the document, as illustrated in Table 3 and Table 4. (Table 4 gives examples for those inconsistencies that may not seem straightforward from the descriptions in Table 3.) There exist other inconsistencies in the document but they are only due to the fact that the document is not complete: e.g., not all the use cases are described with an interaction diagram, not all the classes have attributes.

One inconsistency is particularly more complex than all the others as it involves an interaction diagram, two classes (and their invariants), a public method and a constructor (and their contracts). Its description below is supported by Fig. 2.

Table 3. Taxonomy of inconsistencies

	Total number
In the data dictionary (e.g., erroneous operation description, missing or incorrect contract)	14
Between interaction and class diagrams (e.g., message suggests navigation between objects but there is no association)	2
Between class diagram and data dictionary (e.g., missing operation in data dictionary, inconsistent signatures in class and data dictionary, inconsistent scopes in class and data dictionary, missing attribute description in data dictionary, inconsistent association role names or multiplicities, attribute redundant with association, missing association)	17
Between interaction diagram and data dictionary (e.g., wrong operation used in message, inconsistency of message sequence with operation contracts, wrong sequence of messages, precondition not checked in message sequence, wrong target object type for message)	12
Between state machine diagram and data dictionary (e.g., missing state, missing transition, missing guard condition)	12

Table 4. Illustrating some inconsistencies

Inconsistency	**Example**
Erroneous operation description	In operation `Cab::assignCab(j:PrebookedJob)`, the textual description should mention pre-booked jobs instead of immediate jobs.
Missing or incorrect contract	The textual description of `JobManagerControl::sendCab()` is not consistent with its post-condition: the description mentions both immediate and pre-booked jobs whereas the post-condition only constrains immediate jobs.
Message suggests navigation between objects but there is no association	A message is sent from an instance of class `ImmediateJob` to an instance of class `NotifyNewJobs` but there is not association (or path) between those two classes in the class diagram.
Inconsistent role names between class diagram and data dictionary	The data dictionary shows an OCL navigation of the form `self.roleName` but the class diagram does not show a role name on any association in which the context class is involved.
Attribute redundant with association	Class `Cab` has an attribute `driver` (of type `Driver`) as well as an association to class `Driver`.
Wrong operation used in message	In a message, operation `disptachCab()` should be used instead of `assignCab()` (the two operations are very similar).
Inconsistency of message sequence with operation contracts	According to pre and post-conditions, all the messages triggered by the constructor of class `ImmediateJob` (in the sequence diagram for use case `DispatchPrebookedJob`) should in fact be called by the caller of that constructor (e.g., the services offered by these messages are not the responsibility of that constructor, as per its post-condition).
Precondition not checked in message sequence	Operation `getFirstCab()` has a precondition (`cab.allInstances->size >= 1`) that is not checked before its use in a message.

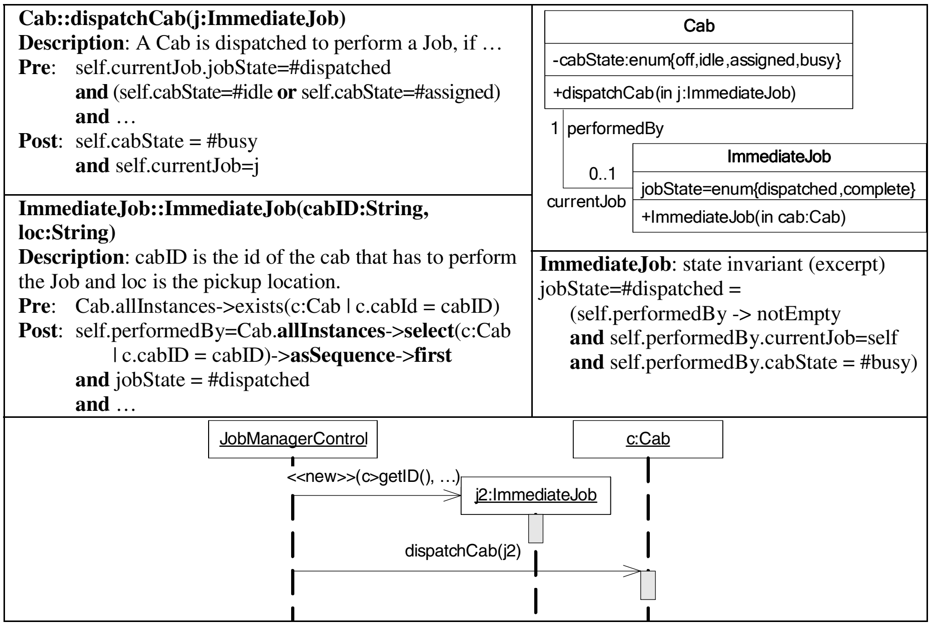
The signature of `Cab::dispatchCab(j:ImmediateJob)`, its description, and its post-condition suggest that its responsibility is to set this cab's `currentJob` link to `j` (the parameter). However, the precondition indicates that this link is assumed to already exist (reference to `self.currentJob`). So if the intent (i.e., post-condition) of the operation is correct, the precondition should read `j.jobState = #dispatched` instead of `self.currentJob.jobState = #dispatched`.

Additionally, the postcondition of `ImmediateJob`'s constructor indicates that the state of the `ImmediateJob`, once created, is `#dispatched`. According to the state invariant, this means that once the constructor finishes, the link from the `ImmediateJob` object to the `Cab` object should be set (`self.performedBy->notEmpty`) and the link from the `Cab` object to the `ImmediateJob` object should be set too (`self.performedBy.currentJob = self`). In other words, the purpose of the constructor is to set the links in both directions.

However, the interaction diagram shows that calls to the `ImmediateJob` constructor and operation `dispatchCab()` are performed in sequence by another object. So the interaction diagram suggests that the link is set twice.

Additionally, the `ImmediateJob`'s invariant specifies that the `Cab` state is `busy`, whereas the precondition of `dispatchCab()` (which immediately follows the call to the constructor of `ImmediateJob`) requires that this state be `idle` or `assigned`.

There is therefore a contradiction: setting the link between the `ImmediateJob` and the `Cab` objects should be the responsibility of either the constructor of `ImmediateJob` or `Cab`'s operation `dispatchCab()`, but not both, and the pre and post conditions and invariants should be consistent with the ordering of messages in the interaction diagram.



Cab::dispatchCab(j:ImmediateJob)
Description: A Cab is dispatched to perform a Job, if …
Pre: self.currentJob.jobState=#dispatched
and (self.cabState=#idle **or** self.cabState=#assigned)
and …
Post: self.cabState = #busy
and self.currentJob=j

ImmediateJob::ImmediateJob(cabID:String, loc:String)
Description: cabID is the id of the cab that has to perform the Job and loc is the pickup location.
Pre: Cab.allInstances->exists(c:Cab | c.cabId = cabID)
Post: self.performedBy=Cab.**allInstances**->**select**(c:Cab | c.cabID = cabID)->**asSequence**->**first**
and jobState = #dispatched
and …

ImmediateJob: state invariant (excerpt)
jobState=#dispatched =
(self.performedBy -> notEmpty
and self.performedBy.currentJob=self
and self.performedBy.cabState = #busy)

Cab
-cabState:enum{off,idle,assigned,busy}
+dispatchCab(in j:ImmediateJob)
1
performedBy
0..1
currentJob
ImmediateJob
jobState=enum{dispatched,complete}
+ImmediateJob(in cab:Cab)
JobManagerControl
c:Cab
<<new>>(c>getID(), …)
j2:ImmediateJob
dispatchCab(j2)

Fig. 2. Example of complex inconsistency (excerpts)

4 Lessons Learned

The discussion of this section is the result of our observations of laboratory results of, and comments made by, the 437 students who have been following the Software Engineering course and who therefore attended this first laboratory exercise on UML diagram consistency rules since 2003.

Overall, the students were not able to find all the seeded inconsistencies. On average, students were only able to find 11.5 inconsistencies; The minimum and maximum number of inconsistencies found by a student are 6 and 21, respectively. Most of the identified inconsistencies were the ones we seeded between the class diagram and the data dictionary. Our explanation is that (most of) those inconsistencies are simple to find as they mostly consist in different writings (or spelling) of the same information: e.g., inconsistent operation signatures, wrong multiplicities; and do not require a deep understanding of the UML semantics to be caught (they could likely be caught automatically by a tool). (Automated identifications of those inconsistencies can therefore be investigated.) Most of the other inconsistencies being reported by students involve interaction diagrams and the data dictionary: especially the consistency of message sequences and operations contracts. This is probably due to the fact that we discussed an example of such an inconsistency during the introductory lecture in detail. (Automated identification of those inconsistencies can also be investigated, though they may require complex manipulations and evaluations of OCL expressions.) Only a few students were able to discover the more difficult inconsistency discussed previously in Section 3.3. (Although automatically identifying this problem can be further investigated, we have doubts that this is feasible as finding it requires a deep understanding of the model.)

Table 5. An informal description of consistency rules

	Use case diagram	Use case description	Interaction diagram	State Machine diagram	Class diagram	Data dictionary
Use case diagram	NA	Each use case in the use case diagram is described by a use case description	Each use case is accompanied by an interaction diagram			Actors and use cases appear in the data dictionary
Use case description	Each use case description corresponds to a use case in the use case diagram	NA	Services described in each use case description find a realization in an interaction diagram		Data manipulated by the system (as specified in use case descriptions) appear as Entity classes in the class diagram	Actors and use cases appear in the data dictionary
Interaction diagram	Each interaction diagram corresponds to a use case in the use case diagram	The flow of messages in an interaction diagram corresponds to the flow of steps in the corresponding use case description	NA	State changes suggested by scenarios in an interaction diagram are legal as per a state machine diagram	Objects in an interaction diagram are instances of classes in the class diagram. Types in an interaction diagram appear in a class diagram. Message signatures refer to operations or (signal) classes in a class diagrams	Scenarios in interaction diagrams are coherent with contracts (i.e., operation pre and postconditions) of the data dictionary
State Machine diagram			All legal (sequence of) state changes in a state machine diagram matches a (sequence of) messages	NA	Operations (i.e., actions) in the state machine diagram appear in a class diagram	States in the state machine diagram are described with state invariants in the data dictionary
Class diagram		Entity classes correspond to data manipulated by the system as described in a use case description	Classes and operations are used in the set of interaction diagrams	Classes and operations are used in the set of state machine diagrams	NA	Classes, attributes, operations and contracts (class invariants, operation pre and post conditions) are described in the data dictionary
Data dictionary	Describes use cases and actors in the use case diagram	Describes use cases and actors in use case descriptions	Contracts are coherent with scenarios in interaction diagrams	State invariants describe states in state machine diagrams	Describes classes, operations and attributes of the class diagram	NA

There are probably different reasons for the small number of reported inconsistencies. Among others, we can mention:

1. Although introduced to the UML notation during the pre-requisite "Systems Analysis and Design" course, this was the first time the students were facing a fairly large and complex analysis document for a real system;

2. The document we used was perhaps too large. A reduced document that would focus only on the main use cases (e.g., the three use cases for which we have sequence diagrams), that would show in the class diagram only the classes, attributes and operations that are used in the three interaction and three state machine diagrams, would be more manageable;
3. The laboratory session was only three hours long and half an hour was dedicated to the introductory lecture: only two and a half hours remained for reading, understanding the 40-page long analysis document and finding inconsistencies. The students therefore had to focus on some parts of the document and ignore others;
4. The students were not instructed to follow a specific inspection technique for UML documents, though such techniques have been proven to be effective at finding problems (see for instance [4, 13, 14]). Instead they followed a heuristic to traverse the analysis document. An inspection technique could have produced better results.

The laboratory exercise however achieved its objectives since:

a. The students were exposed for the first time to a reasonably complex analysis document involving the main UML diagrams;
b. They were able to find inconsistencies, thereby showing that they actually went through the document and understood the analysis diagrams. Notably, a very large number of students were able to find at least one inconsistency involving a message sequence and operations contracts;
c. They learnt that the UML is more than annotated boxes and lines and that design decisions made in one diagram have impacts on other diagrams.

Overall, student feedback was very positive. Although no questionnaire was used to get student feedback, informal discussions at the end of the laboratory session and later during the term indicated that students did get the message. Quoting a student: “Before that laboratory session, I did not have a clear understanding of what the UML was all about.” Additionally, the instructor of the Analysis and Design Laboratory course, that follows the course in which the laboratory session took place, reported that she noticed a clear increase in the performance and the UML proficiency of the students since the introduction of the laboratory session.

5 Conclusion

The Unified Modeling Language (UML) [11] is now the de-facto standard for the analysis and design of object-oriented software systems [12]. Though the UML has semantics, in particular under the form of well-formedness rules [11], and though these rules have not been recently introduced (many were already part of the UML 1.3 standard in 1999), many practitioners and students do not seem to be aware of basic UML well-formedness rules. This is part of the broader problem of the consistency between views in a UML analysis/design document: the different views each focus on one aspect of the software, but together describe the whole software and should therefore be coherent. As a result practitioners perceive the UML as a graphic tool that merely allows them to draw annotated boxes and lines freely.

In this article, we reported on an attempt we made to heighten student awareness of the problem of the well-formedness (and more broadly, consistency) of UML diagrams. We described the laboratory material we used to that end and reported on our experience of its use since 2003. Experience has shown that this laboratory material was indeed useful, and that specific training in what the semantics of the UML are (e.g., by identifying inconsistencies) improves students' ability to apply the language.

Future work will include improving the laboratory material to include other kinds of well-formedness (or consistency) rules, restructuring the analysis document to allow students to discover more inconsistencies, and trying to provide automated tool support for as many rules as possible.

References

[1] Berenbach, B.: The Evaluation of Large, Complex UML Analysis and Design Models. In: Proc. ACM International Conference on Software Engineering, pp. 232–241 (2004)

[2] Bontemps, Y., Heymans, P., Schobbens, P.Y.: From Live Sequence Charts to State Machines and Back: A guided Tour. TSE 31(12), 999–1014 (2005)

[3] Bruegge, B., Dutoit, A.H.: Object-Oriented Software Engineering Using UML, Patterns, and Java, 2nd edn. Prentice Hall, Englewood Cliffs (2004)

[4] Conradi, R., Mohagheghi, P., Arif, T., Hegde, L.C., Bunde, G.A., Pedersen, A.: Object-Oriented Reading Techniques for Inspection of UML Models – An Industrial Experiment. In: Cardelli, L. (ed.) ECOOP 2003. LNCS, vol. 2743, pp. 483–501. Springer, Heidelberg (2003)

[5] Doolan, E.P.: Experience with Fagan's inspection method. Software Practice and Experience 22(2), 173–182 (1992)

[6] Fagan, M.E.: Design and code inspections to reduce errors in program development. IBM Systems Journal 15(3), 182–211 (1976)

[7] Kleppe, A., Warmer, J., Bast, W.: MDA Explained - The Model Driven Architecture: Practice and Promise. Addison-Wesley, Reading (2003)

[8] Lano, K.: A Compositional Semantics of UML-RSDS Software and Systems Modeling (to appear)

[9] Meyer, B.: Design by Contracts. IEEE Computer 25(10), 40–52 (1992)

[10] OMG, OCL 2.0 Specification. Object Management Group, Final Adopted Specification ptc/03-10-14 (2003)

[11] OMG, UML 2.0 Superstructure Specification. Object Management Group, Final Adopted Specification ptc/03-08-02 (2003)

[12] Pender, T.: UML Bible. Wiley, Chichester (2003)

[13] Shepard, T., Kelly, D., Smith, R., Chisholm, R., Jackson, T., Mondoux, P.: Inspecting designs in the context of model-driven development. In: Proc. Conference of the Center for Advanced Studies on Collaborative Research (2006)

[14] Travassos, G.H., Shull, F., Carver, J.: A Family of Reading Techniques for OO Design Inspections. In: Proc. Brazilian Symposium on Software Engineering: Workshop on Software Quality (2000)

[15] Ziadi, T., Helouet, L., Jezequel, J.M.: Revisiting Statechart Synthesis with an Algebraic Approach. In: Proc. ICSE 2004, pp. 242–251 (2004)

Automatic Checklist Generation for the Assessment of UML Models

Tom Gelhausen, Mathias Landhäußer, and Sven J. Körner

Institute for Program Structures and Data Organization
University of Karlsruhe
76131 Karlsruhe, Germany
{gelhausen,lama,koerner}@ipd.uni-karlsruhe.de

Abstract. Assessing numerous models from students in written exams or homework is an exhausting task. We present an approach for a fair and transparent assessment of the completeness of models according to a natural language domain description. The assessment is based on checklists generated by the tool SUMOχ. SUMOχ directly works on an annotated version of the original exam text, so no 'gold standard' is needed.

1 Introduction

If we teach modeling we also need to grade models. But the task of modeling comprises a certain degree of freedom, and this renders exam questions on that task somewhat awkward: Assessments are frequently rejected by the examinees. This is either due to mistakes of the examiners or due to unreasonableness of the examinees. No doubt, the examiners make mistakes as the task of assessing numerous models is exhausting, requires fast perception, and great mental flexibility. But as frequently, students tend to challenge assessments on a pursuit for a better grade. Experience shows, that especially modeling tasks lend themselves to discussing every single detail of an assessment in terms of correctness and adequacy. Instant insight on the students' side seems aspirational to solve both problems: a) discover true errors made by the examiners and b) avoid exhausting discussions during homework reviews and post-exam reviews.

We identified the need for an evaluation procedure that meets the following requirements: The assessments should be systematic, fair, transparent, and *cumulative* in a sense that they consist of simple atomic decisions which themselves are (ideally) unquestionable. An assessment based on a sample solution is problematic, as one tends to use this *sample* as a yardstick for all of the students' solutions. We discourage the use of such a 'gold standard', since it stultifies the examiner's perceptivity. Our approach uses comprehensive and self-explanatory checklists to assess UML models[1]. Based on a technique which we initially designed to automatically generate domain models [1], we developed

[1] Please note that the approach is not limited to class diagrams. We currently support class and sequence diagrams, state charts and OCL.

Sumoχ (Sal$_E$-based UML MOdel eXamination). It works on a semantic annotation of the original exam question's text and is therefore able to generate considerably less biased checklists than one would create by hand.

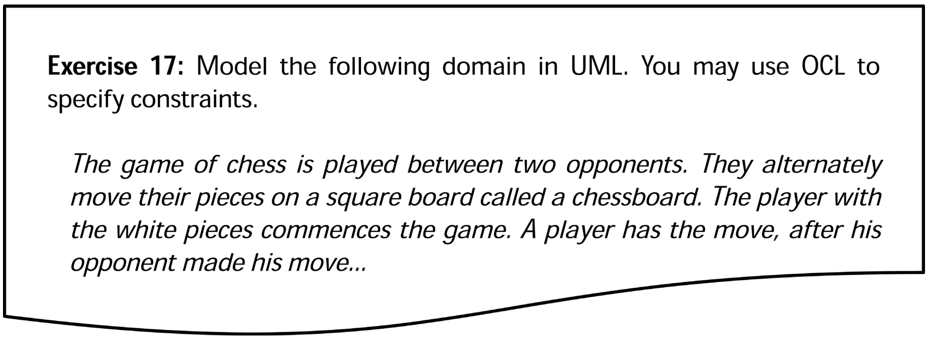
The Sumoχ-checklists are designed to measure the *completeness of the content* of a model according to a given domain description. Established metrics on models measure quality aspects such as similarity and class relationship complexity (cf. Section 5). In this respect, general checklists on model quality and the domain-dependent Sumoχ-checklists complement each other.

We start with the explanation of the approach and how it can be used to generate checklists. We then evaluate the complete process using the checklists to assess students' homework and exams. The lessons we learned during the evaluation are discussed afterwards. The following section focuses on the related work and also serves as foundation for our idea since it shows the necessity of such a method. The article closes with a conclusion.

2 The Approach

Assume you want to give your students a modeling task as it is depicted in Figure 1. You provide a text describing the application domain and you expect your students to model it concisely regarding this text.

We propose to use a systematic checklist to grade the results from the students. Table 1 shows an excerpt of such a checklist for the text above. This checklist was generated by Sumoχ and has already been filled out by an imaginary examiner (in script style). It contains one section per sentence of the original domain description. For example, row 1 introduces the first section of this checklist. Each section lists all elements that could be modeled. For the first section, *opponents* and the *Multiplicity 2* are exemplified (rows 2-6 and 7-10). The *opponents* in turn give an example for a linguistic structure, a constituent in this case, that can be modeled in various ways: They could be modeled as a class (row 2) or as a role (-name) within a UML association (row 3). Sumoχ also proposes that *opponents* could be modeled as an instance of another class (row 4) – possibly meaningless *in this case* but obviously meaningful in the case of the element



Exercise 17: Model the following domain in UML. You may use OCL to specify constraints.

The game of chess is played between two opponents. They alternately move their pieces on a square board called a chessboard. The player with the white pieces commences the game. A player has the move, after his opponent made his move...

Fig. 1. Example modeling task

Table 1. Example of an Assessment Based on a SUMOχ-Questionnaire

Row	**Element of Phrase**	**Modeled...**	**yes**	**no**	**wrong**
1	**Phrase: The game of chess is played between two opponents.**				
2	"opponents"	as class		✗	
3		as role		✗	
4		as instance		✗	
5	Differently (please specify):				
6	Explicitly not modeled (please specify): Opponent- and Player-classes are equivalent		✗		
7	Multiplicity "2" of "opponents"	as multiplicity		✗	
8		in an OCL constraint			✗
9	Differently (please specify):				
10	Explicitly not modeled (please specify):				
...	*etc.*				
27	**Phrase: The player with the white pieces commences the game.**				
28	"player"	as class	✗		
29		as role		✗	
30		as instance	✗		
31	Differently (please specify):				
32	Explicitly not modeled (please specify):				
33	Attribute "white" of "pieces"	as Boolean function (with a parameter)	✗		
34		as scalar function		✗	
35		as state of "pieces"		✗	
36		as attribute of "pieces"	✗		
37	Differently (please specify):				
38	Explicitly not modeled (please specify):				
...	*etc.*				

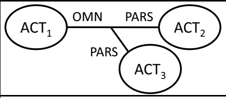
player below (row 30). However, our imaginary examiner did not find a class, a role, or an instance that covered the concept of *opponents*. Instead, he found a note from the student that the classes *Player* and *Opponent* were unified (row 6).

Obviously, our approach maps linguistic structures[2] to model elements. This approach originated in Abbott's 1983 article "Program Design by Informal English Descriptions" [2]. Our method incorporates various suggestions that have been published in the domain of (automatic) model extraction since then (see [3] for a survey of approaches). We collected all UML fragments we found that were generated from single (simple or complex) linguistic structures; Table 2 enumerates some of the resulting mappings. Up to now, we identified a total of 35 rules[3] to turn linguistic structures into questions about model elements. Applying these rules and generating the appropriate questions can be done by hand, of course. Yet, we created SUMOχ to apply these rules automatically and published an online version of the tool together with the ruleset [4].

[2] On a syntactic level such structures are expressed via noun phrases or adverbials, for instance. But actually, we do not map *syntactic* patterns on the text source, but rather *semantic patterns* on the internal discourse model (cf. Table 2).

[3] The exact number significantly depends on the intelligence of the rule processor. The number 35 would apply to a 'human processor'. A computer program is less fault-tolerant according to the applicability of the single rules and thus requires (in our case) a total of 120 GrGen.NET-rules for all variations.

Table 2. Linguistic Structures matched to UML (excerpt)

Linguistic Structure	**Explanation**	**UML model element**
object with the role **AG**	An acting person or thing executing an action	Class, role, or instance
object with the role **PAT**	Person or thing affected by an action or on which an action is being performed	Class, role, or instance
object with the role **ACT** (+AG +PAT)	An action, executed by AG on PAT, or a relation between AG and PAT	Method named ACT at the AG class with PAT param, association named ACT between AG and PAT classes, state or transition named ACT, etc.
	A complex action (OMN) composed of multiple other actions ($n\times$PARS).	Subcall of ACT$_2$ and ACT$_3$ from ACT$_1$ in sequence diagram or in a comment on method ACT$_1$
	etc.	

Listing 1. SAL$_E$-Annotated Domain Description

```
[ #The game_of_chess|PAT #is played|ACT #between *two opponents|AG ].

[ They|AG move|ACT their|POSS pieces|{HAB,PAT} $alternately<<3 #on
  [ #a $square ^board|{PAT,FIN} called|ACT #a chessboard|FIC ]|LOC_POS ].
[ @They|EQD @opponents|EQK ].
[ @their|EQD @opponents|EQK ].

[ [ #The ^player|POSS #with #the $white pieces|HAB ]|AG
  commences|ACT #the game|PAT ].
[ @game|EQD @game_of_chess|EQK ].
```

2.1 Preparing the Text

To use SUMOχ, one first annotates the domain description using SAL$_E$. This is an annotation language for the semantic markup of natural language [1]. This task does not require any modeling knowledge, since it is just a linguistic annotation. Neither does it require any deeper grammatical knowledge (even though this might help), as the annotation concerns semantics, not syntax. The annotated result for the above example is shown in Listing 1. The original text is denoted in bold font whereas all added information is non-bold.

Roughly speaking, SAL$_E$ distinguishes 'objects', n-ary relations, and the roles that the objects play in these relations. As you can see in Listing 1, all clauses and all sub-clauses are embraced using square brackets (`[]`). Each pair of brackets denotes a relation. Within a relation, SAL$_E$ treats single words and nested relations as units. A word is commented out with a hash prefix (`#`). An 'object' is marked by denoting its role(s) using a vertical bar (`|`) in conjunction with the role name(s). Examples for roles used in this excerpt are AG, ACT, and PAT.

These three roles are also described in Table 2. (For a complete list of the 44 roles our system currently supports, see [1] or [5].)

In Sal$_E$, a nested relation (like in line four: *a square board called a chessboard*) can per se take a role in its outer relation. Yet, if an inner object of the nested relation is marked with a caret (^), it is interpreted as the head of the clause (like in ANTLR grammars). This head is then bound to the outer relation with the role which is appended to the nested relation. Thus the *board* in line four takes the roles PAT and FIN in the inner relation and the role LOC_POS in the outer relation.

Each object can be attributed in Sal$_E$. An attribute is denoted with a dollar sign ($) and applies to the following object – except when it is explicitly reassigned using the movement operators (<< and >>): The *alternately* in line three is reassigned to modify the *move*-action this way. A special form of an attribute is multiplicity which is denoted using an asterisk (*).

As Sal$_E$ is rather a compiler than a sophisticated natural language processor (cf. Section 2.2), we need a simple and deterministic way to cope with coreferences[4]. There are two ways in Sal$_E$ to express coreferences: either via word occurrence references or via assertions. A word occurrence reference is denoted with an at-sign (@) in front of a word. It tells the system, that a word refers to the very same thing (same instance) as in the preceding occurrence of that word – not just to "any instance of this category", which is the default. The lines five, six, and ten contain assertions, expressed via relations with special roles. These relations do not directly originate from the text. The annotator has entered them to tell the system that, for instance, *they* and *opponents* means the same thing. The role EQK denotes the one of the two *eq*ual concepts that should be *k*ept in the internal discourse model, while EQD denotes the one that should be *d*ropped. The object that has been annotated with EQK inherits all roles and attributions from the dropped object in this case. For a more detailed description of this process see [5].

2.2 Implementation

The questions are generated based on pattern matching on the discourse model. Therefore, Sumoχ uses the Sal$_E$ compiler to obtain this discourse model from the annotated text. Technically, the Sal$_E$ compiler is based on an ANTLR [6] generated parser, thus avoiding resource consuming and error-prone Natural Language Processing (NLP). It builds a graph representation of the discourse for the graph rewrite system GrGen.NET [7]. The graph rewrite system then executes the above-mentioned 35 (respectively 120) declarative transformation rules. Finally, the surface structures (i. e. the actual questionnaire) are generated via the application of XSL-T templates.

An optimization that we implemented in the rule set for GrGen.NET concerns the repetition and the order of the generated questions: The rules do not produce more than one question about elements that are mentioned in the domain

[4] Coreference: two or more expressions refer to the same extra-linguistic entity.

description multiple times. Because of this, the list contains questions about the existence of elements (for example classes) in the beginning, and questions about their behavior (methods, state transitions) or their design (attributes, associations) later on. This eases the assessment since the examiner first needs to find the element before he can answer questions about it. However, some manual rework of the resulting lists is necessary, since the generation is simple and straightforward. In our experience, the manual rework is largely limited to deleting superfluous questions and adapting flections. We regard this as a negligible issue as a user of Sumoχ will probably revise the list for his own version anyway, for example including model quality criteria.

3 Evaluation

We evaluated our approach in various ways. First of all, we successfully used it for grading exams and homework – and plan to do so in the future. For exams, we determined student complaint rates. We compared these rates with historical data on non-modeling questions. The results suggest that assessments created this way are acceptable to the students. Yet, as the historical data on exam results has not been collected with this research in mind, we currently cannot conduct other studies on exam results and the students' complaint behavior. To further evaluate our approach, we conducted an extensive study on a large homework in which we asked pairs of students to model the game of chess. We provide statistical analyses of the results which show a high inter-rater agreement. Besides the numerical results, we examined the discrepancies of the examiners' ratings on a qualitative basis. We present our insights in Section 4.

3.1 Legitimate Complaints in Exams: 'Intra-Student' Fairness

Written exams for our software engineering lecture are held twice a year and take 60 minutes. Typically 60 to 250 graduate students of computer science attend the lecture and take the exam. After the assessment, we hold a post-exam review in which the students can double-check. If they find grading errors, they can ask for correction. After this review, the final grade is determined. Approximately 45 % of the examinees attend these reviews.

Two recent exams included a modeling task worth 17 of 60 points. We explicitly advised our students to examine the text carefully and to closely analyze every part of every phrase on its own. Since they were pinched for time, we did not expect their models to be overly sophisticated, but that they kept closely with the domain description.

We compared the results with the results of 22 non-modeling questions from written exams over the last three years. These 22 questions are of well established types, for example memorization questions such as 'Name three stages of the waterfall model'. These types can be assessed in a fair and transparent manner and lead to few legitimate complaints in the post-exam reviews. In the two modeling tasks, the students achieved moderate results, so complaints were to be

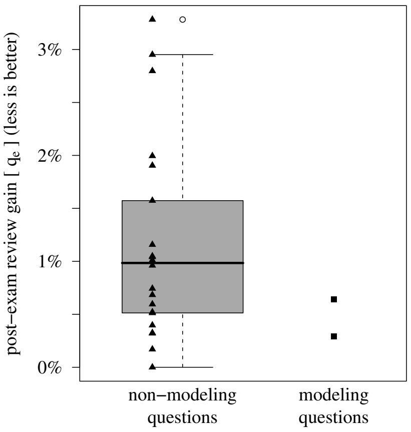


Fig. 2. Legitimate complaints about assessments (post-exam reviews)

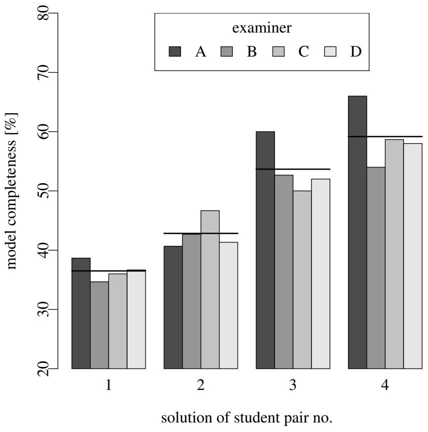


Fig. 3. Comparison of ratings (assessed homework)

expected. During the post-exam reviews, we handed out the filled-out checklists along with the exams. This way, the ratings were completely transparent to the students. A low number of legitimate complaints would be considered fair for each individual student[5].

To compare the assessments of the questions, we computed the post-exam review gain q_e as the mean difference between the results before (p^{pre}) and after (p^{post}) the post-exam review for every exercise e for all students s that reviewed their exam (let n be the number of these students). We normalize the post-exam review gain by the maximum number of points p_e^{max}, that the students could obtain in the corresponding exercise: $q_e := \frac{1}{n}\sum_{s=1}^{n} \frac{p_{es}^{post} - p_{es}^{pre}}{p_e^{max}}$. The results are depicted in Figure 2: The post-exam review gain of the 22 non-modeling questions (left side) are opposed to the two modeling tasks (right side) in this diagram. Every triangle and both squares represent the post-exam review gain for one exam question. The average n (i. e. the number of students that reviewed their exam in one of the six review sessions under scope) is 64.8. The results suggest that our way of assessing UML models can keep up with other (evolved) types of exam questions: More than half of the 22 non-modeling questions performed worse in terms of the post-exam review gain.

3.2 Homework Study: 'Inter-Student' Fairness

Our experience from post-exam reviews shows that assessments are repeatedly biased by the examiners' different levels of generosity (in the sense of their margin of discretion). Nonetheless, an assessment method should still be monotonic in a sense that better students always get better grades. We conducted a study where

[5] We suspect that in this case every student feels the assessment of *his* work to be profound. We examine 'inter-student' fairness in Section 3.2.

Table 3. Pairwise correlation with Pearson's correlation coefficient

(r,p)	examiner A	examiner B	examiner C	examiner D
examiner B	(0.904,4.8%)	–	–	–
examiner C	(0.826,8.7%)	(0.935,3.2%)	–	–
examiner D	(0.964,1.8%)	(0.968,1.6%)	(0.945,0.3%)	–
avg	(0.961,1.9%)	(0.976,1.2%)	(0.945,2.7%)	(0.999,0%)

multiple examiners independently assessed a randomly chosen set of student homework. A high inter-rater agreement would indicate that the students rank independently from the examiners. This study was run to verify this property in order to ensure inter-student fairness.

We chose the FIDE laws of chess as domain description to be modeled in UML. This resulted in a domain description of four pages. The students attended the software engineering lecture and were mainly graduate students of computer science. They were allowed to work in pairs and had to develop a UML model accompanied with OCL as part of an assessed homework. For this study, we randomly picked four student pairs and asked four examiners from the staff of our chair to rate their work. The diagram in Figure 3 shows the evaluated model completeness of the inspected UML models. The horizontal line indicates the examiner groups' average rating for each model.

In order to compare the agreement of every possible examiner pair, we computed Pearson's correlation coefficient[6] r (for details see [8]) of each examiner with every other. Since we expect the results of the examiners to be positively correlated, we try to reject the null hypothesis $H_0 : r \leq 0$. We assume the null hypothesis to be rejected if the p value calculated with a one-tailed t-Test is below the significance level[7] of $\alpha = 0.05$.

The first three rows of Table 3 show the correlations among the examiners. The values of r suggest that there is a strong linear relationship between the results of the examiners. Only the p value of the correlation between examiner A and C is greater than the significance level α, but the probability that the corresponding correlation coefficient is due to chance is still below 9 %. Even though this value does not allow us to reject H_0 on a significance level of α, we support the conclusion that our approach is indeed successful.

To investigate the consistency of the examiners' rating with the average group rating, we calculated Pearson's correlation coefficient of every examiner with

[6] The values of r can range from -1 to $+1$ and indicate strength and direction of a linear relationship between two variables X and Y, in our case of the scores for the models of two examiners. A value of 0 means that we have no linear relationship between the variables. Positive values indicate a positive linear relationships, i. e. whenever the value of X increases, the value of Y increases, too. Negative values indicate a negative linear relationships, i. e. whenever X increases, Y decreases.

[7] Meaning the probability that our observed correlation coefficients r (or greater coefficients) could be due to chance is less than 5 %.

the group average. The resulting correlation coefficients and their corresponding p values can be found in the last row of Table 3. Again we found high values of r suggesting a strong linear relationship between the ratings of the examiners and the group's average rating. This time, all r values are statistically significant. This clearly states that the decision of a single examiner is consistent with the group's decision.

4 Observations and Discussion

As our homework study showed (cf. Figure 3), we have not reached absolute unquestionableness yet. Comparing the different examiners' results, we observed that they are still biased – so there still remains room for interpretations. This does not impose a threat to assessments conducted by a single examiner (or a small group of examiners who synchronize their decisions): In our study, the examiners were strictly isolated to ensure statistical independence. Yet, after the assessments, the examiners discussed the discrepancies in their decisions.

4.1 Observations Concerning the Examiners

Examiners had different interpretations of the models and the meaning of some list items. We need to minimize this margin of discretion. An example is the rook's multiplicity which could have been 2, [0..2], or [0..10]. (The latter is possible through pawn-promotion.)

After a while, the examiners tended to mark list items which were clearly non-existent in the students' models. This could be due to the fact that only having parts of the model sometimes resulted the examiners forming their own – more complete – model in their mind. For example some examiners marked classes as modeled though they were only *indicated* in a state chart.

A huge effort is finding the sought-after item, especially if the models comprise multiple pages (as in the case of our homework study). UML diagrams in digital form could ease this effort as computer-based find functions could speed-up the process. Exams might still be written by hand, but don't usually exceed a certain model size due to time constraints.

It is also hard to ensure that examiners find and check all possible versions: If an element occurs multiple times in the model, it also has to be marked multiple times in the list in our study. But the examiners tended to stop searching after an element was found for the first time.

4.2 Observations Concerning the Students

Some students delivered models that were hardly UML. Collectively using a UML tool should help to check the models for compliance with the rules of UML, thus easing the assessment.

A number of students handed in code rather than models. This contradicts the idea of abstraction and rather gives a concrete implementation. This misunderstanding can be prevented by clear formulation of exam questions.

Some students used domain knowledge to complement their models: They added artifacts which were not given in the domain description. In turn, some students left out parts that were in the description, for example the (rather complex) castling move. We conclude that some of the students did not focus on the provided text, but rather modeled the game by what they knew by heart. Our approach of making content-completeness the primary basis of valuation leads to bad grades for these students. The examiner must decide whether this poses a threat to the validity of his ratings.

4.3 Applicability

The process we presented here still depends on the provided text to a large extend. In comparison to many other model extraction approaches (see [3], for example) $\text{S}\textsc{al}_E$ is much less dependent on the chosen formulation. Nevertheless, our approach can only work with content that a) actually resides in the provided text and that b) is also annotated correctly.

The effort for the annotation depends on the experience of the annotator using $\text{S}\textsc{al}_E$, but it also depends on the given domain description. This effort may also tip the scales regarding the applicability of our approach as a metrics in industry. (Indeed, there seems to be a need for a metrics for content-completeness of models, as we show in the next section.) On the one hand, the effort in learning $\text{S}\textsc{al}_E$ mainly comes from learning and applying the 44 roles we currently support. Practice will show whether and how this role set can be optimized in the future. On the other hand, effort originates from 'linguistic defects' in the domain description. Linguistic defects are unobvious semantic or pragmatic flaws in the text, such as deletions, generalizations, and distortions (see [9] for details). Generally, it seems that 'better' specifications are easier to annotate.

Concerning the grading of models, the approach is limited in two ways: First, it is not suited to grade (object-oriented) analyses, it has been designed to grade models of a certain, precisely prescribed domain. Second, it is not suited to assess architectural decisions that might be contained in a model. Both limitations are by design – even though we support the opinion, that analysis and architecture are two important subtasks of modeling in practice.

Regarding the first limitation (grading analyses), we understand that this is a challenging but particularly a *different* task. Technically, it could be done with our approach by assigning the task to the students withholding certain aspects (i. e. sentences) from the domain description. The philosophical question behind this approach is how fair it is: The examinee not only has to catch (guess) all of the expected aspects, he or she also has to guess the intended degree of relevance for the model. We have no idea, whether (and how) the omitted aspects can be made "obvious enough".

Concerning the second limitation (assessing architectural decisions), there are already plenty of metrics available. These can be used to complement our approach without difficulties.

5 Related Work

Grading models is exhausting as there are usually countless accurate solutions: Lange et al. [10] show that UML models bring the freedom of creativity to the modeler because they can be ambiguous *and* correct at the same time. The lack of uniformity and the large amount of defects contained in UML models result in miscommunication among UML readers. Lange et al. [10] propose modeling conventions, analogous to coding conventions for programming. Results indicate that decreased defect density is attainable at the cost of increased effort when using modeling conventions. This shows that assessing UML models remains a complicated – and not yet mastered – task. Modeling conventions could probably be enforced by tailoring our assessment questions to the proper settings.

Lange and Chaudron show that modelers who tend to exploit the freedom of UML impose risks on software projects [11] and their modeling phase [12]. One has to take into account that even unambiguous and syntactically correct models can be incomplete with regard to the users' needs and specification [13].

Mohagheghi and Aagedal [14] agree with our basic assumption that model content-completeness is important. They ran a general survey of various methods on determining model quality. As it turns out, quality assurance techniques must not only consider the quality itself but also the completeness of UML models. The earlier one focuses on model completeness, the better.

In 2006, Lange and Chaudron [15] conducted a survey among practitioners which analyzed UML models in 14 case studies. Even though 52 % of the practitioners responded that model completeness is a major stopping criteria for their modeling activities, there are hardly any metrics to measure completeness. Incompleteness is a big problem: More than half of the reported miscommunication between project stakeholders in subsequent software process stages is due to incomplete models. Stakeholders would like to use metrics two to four times more often than they actually do. Briand et al. [16] point out that working with complete models progresses faster and delivers higher quality software. This shows a need for assessing the completeness of models – even in industry.

Being able to judge the quality of a UML model is important to software quality. No known approach considers checking the models against the textual domain description. But these models are eventually used to create the software. Being the prime artifacts of Model Driven Engineering, models are highly important for the quality of the resulting product [14]. But how can a model be evaluated?

There are some software metrics which are applicable to UML models [17], but most of them are made for analyzing source code. These metrics are usually computed in the last stages of the software process – stages in which a metric cannot show the deficiencies of the product; only the defects of a potentially incomplete implementation. Therefore the model's quality should be evaluated long before the resulting source code itself is created. But the fact that the models deviate from the initial (customer's) textual specification is not yet dealt with. McQuillan and Power show that we need to ensure the software follows the fundamental ideas of the customer [18]. Our approach makes matching the content of a textual specification to existing models feasible.

6 Conclusion

We presented SUMOχ, an approach to automatically generate comprehensive and self-explanatory checklists for the assessment of models. These checklists are intended to grade the content-completeness of models according to a given domain description. They could also serve as a new metrics for the content-completeness of models. A great advantage of our approach is that it does not use a 'gold standard'. Instead, it creates checklists that do systematically *not* restrict the freedom of the modelers. We reached our major design goals of being systematic, cumulative, transparent, and fair. Yet, our studies showed, that there is still room for interpretation while working with the checklists. This leaves room for improvement in future work. Besides an optimization of the role set, tool support for SAL$_E$ could ease the annotation process.

References

1. Gelhausen, T., Tichy, W.F.: Thematic Role based Generation of UML Models from Real World Requirements. In: First IEEE International Conference on Semantic Computing (ICSC 2007), pp. 282–289 (September 2007)
2. Abbott, R.J.: Program Design by Informal English Descriptions. Communnications of the ACM 26(11), 882–894 (1983)
3. Moreno, A.M.: Object Oriented Analysis from Textual Specification. In: Ninth International Conference on Software Engineering and Knowledge Engineering, Madrid, Spain (June 1997)
4. Landhäußer, M.: SUMOχ – SAL$_E$-based UML MOdel eXamination (August 2008), `http://www.ipd.uka.de/~gelhausen/sumox/`
5. Landhäußer, M.: Automatische Erzeugung von Prüflisten zur spezifikationsbezogenen Beurteilung der Vollständigkeit von UML-Modellen. Studienarbeit, Institute for Program Structures and Data Organization (IPD), University of Karlsruhe (July 2008)
6. Parr, T.: ANTLR, `http://www.antlr.org/`
7. Geiß, R., Batz, G.V., Grund, D., Hack, S., Szalkowski, A.: GrGen: A fast SPO-based graph rewriting tool. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 383–397. Springer, Heidelberg (2006)
8. Howell, D.C.: Fundamental Statistics for the Behavioral Sciences, 4th edn. Brooks/Cole Publishing Company (1999)
9. Rupp, C.: Requirements and Psychology. IEEE Software 19(3), 16–18 (2002)
10. Lange, C.F.J., DuBois, B., Chaudron, M.R.V., Demeyer, S.: An Experimental Investigation of UML Modeling Conventions. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 27–41. Springer, Heidelberg (2006)
11. Lange, C.F.J., Chaudron, M.R.V.: Managing Model Quality in UML-Based Software Development. In: Chaudron, M.R.V. (ed.) Proc. 13th IEEE International Workshop on Software Technology and Engineering Practice, pp. 7–16 (2005)
12. Lange, C.F.J., Chaudron, M.R.V.: Effects of Defects in UML models: An Experimental Investigation. In: ICSE 2006: Proceeding of the 28th international conference on Software engineering, pp. 401–411. ACM, New York (2006)

13. Lange, C.F.J., Chaudron, M.R.V.: An Empirical Assessment of Completedness in UML Designs. In: EASE 2004: Proceedings of the 8th Internation Conference on Empirical Assessment in Software Engineering, vol. 920, pp. 111–119 (May 2004)
14. Mohagheghi, P., Aagedal, J.: Evaluating Quality in Model-Driven Engineering. In: MISE 2007: Proceedings of the International Workshop on Modeling in Software Engineering, p. 6. IEEE Computer Society, Washington (2007)
15. Lange, C.F.J., Chaudron, M.R.V., Muskens, J.: In Practice: UML Software Architecture and Design Description. IEEE Software 23(2), 40–46 (2006)
16. Briand, L.C., Hove, S.E., Labiche, Y., Arisholm, E.: Guiding the Application of Design Patterns Based on UML Models. In: Proceedings of IEEE International Conference on Software Maintenance (ICSM), Philadelphia, USA, pp. 234–243 (2006)
17. Kim, H., Boldyreff, C.: Developing Software Metrics Applicable to UML Models. In: Proceedings of QAOOSE 2002 (2002)
18. McQuillan, J.A., Power, J.F.: On the Application of Software Metrics to UML Models. In: Kühne, T. (ed.) MoDELS 2006. LNCS, vol. 4364, pp. 217–226. Springer, Heidelberg (2007)

MODELS Research Projects Symposium

Iulian Ober

University of Toulouse - IRIT,
118 route de Narbonne, F-31062 Toulouse, France
Iulian.Ober@irit.fr

Abstract. This half-day symposium brought together researchers and practitioners around eight presentation of important multi-organization research projects in domains related to the MODELS community.

1 Introduction

The Research Project Symposium was a novelty of MODELS 2008. It aimed to be a showcase for big, multi-organization collaborative research projects in the areas of interest to the MODELS community (software and model engineering at large). The symposium served as a forum for discussing the different organizational models for such projects, to draw conclusions on what works and what does not work well in this type of structure and to derive trends for future topics and collaboration models.

The programme committee selected a wide variety of projects, with respect to their stage (from young to mature and finished projects with well established results), their geography (national/trans-national), the nature of the consortia (academia-only or mixed academia-industry), the funding scheme and agency.

We had 12 submissions, out of which we had to choose only 8 for presentation. The following section makes a brief account of the selected projects.

2 Projects Presented at the Symposium

Modelplex [1]. *Title : MODELling solution for comPLEX software systems* (EU - IST Programme). The project has three major objectives: (1) to develop an open solution for complex systems engineering improving quality and productivity, (2) to lead its industrialisation and (3) to ensure its successful adoption by the industry.

HCDDES [2]. *Title : Frameworks and Tools for High-Confidence Design of Adaptive, Distributed Embedded Control Systems* (USA, multi-university research initiative). The project proposes to develop a comprehensive approach to the design of high-confidence complex embedded systems, by integrating verification, validation, and test procedures throughout the complete design, development and maintenance cycle.

TOPCASED [3]. *Title : Toolkit In OPen source for Critical Applications and SystEms Development* (French DGE project). The project aims at developing an open source CASE environment for critical systems development.

SPICES [4]. *Title : Support for Predictable Integration of mission Critical Embedded Systems* (EU - ITEA Programme). Focusing on aerospace industry SPICES aims at developing an MDD/MDE-compliant tool suite for the design, verification and development of mission-critical RT/E systems dedicated to the aerospace industry.
ALIVE [5]. *Title : A Model Driven approach to Coordination and Organisation for Dynamic Services Engineering* (EU - IST Programme). The project aims to develop new approaches to the engineering of service-oriented systems based on the adaptation of coordination and organisation mechanisms often seen in human and other societies to service-oriented architectures.
FAROS [6]. *Title : Composition Environment for Building Reliable Service-Oriented Architectures* (French ANR project). The project aims at defining a composition environment for building reliable SOAs for applications to be used in ubiquitous environments.
AMPLE [7]. *Title : Supporting Product Line Engineering through Synthesis of Aspect-Oriented and Model-Driven Development* (EU - IST Programme). The project AMPLE (Aspect-oriented, Model-driven, Product Line Engineering) aims at synthesising concepts and techniques from aspect-oriented software development (AOSD) and model-driven engineering (MDE).
Health@net [8]. *Title : Concepts and Prototypes for a Patient-Centric Health Record in Austria* (EU - IST Programme). The project has been part of Austria's preparatory actions with the aim of establishing a shared national electronic health record. It developed concepts and a prototype for a shared patient-centric health record, and allowed to validate the needs of the stakeholders (patients, hospitals, practitioners).

Acknowledgements

The author wishes to thank the other members of the Programme Committee for their help in the selection process: Alain Rossignol (EADS Astrium, France), Andras Pataricza (Budapest University, Hungary), Jochen Malte Küster (IBM, Switzerland), Susanne Graf (VERIMAG, France), Robert France (Colorado State University, USA), Betty Cheng (Michigan State University, USA), Agusti Canals (CS, France).

References

1. MODELPLEX project website, `https://www.modelplex.org/`
2. HCDDES project website, `https://wiki.isis.vanderbilt.edu/hcddes/index.php/Main_Page`
3. TOPCASED project website, `http://www.topcased.org`
4. SPICES project website, `http://www.spices-itea.org`
5. ALIVE project website, `http://www.ist-alive.eu`
6. FAROS project website, `http://www.lifl.fr/faros`
7. AMPLE project website, `http://www.ample-project.net`
8. Health@net project website (in German), `http://healthatnet.at`

Author Index