Centre for Telematics and Information Technology

Secretariat CTIT:
University of Twente
CTIT
P.O. Box 217
7500 AE Enschede
The Netherlands
Phone +31 53 489 3994

ISSN: 0929-0672
CTIT proceedings WP09-12

University of Twente
Enschede - The Netherlands

Fifth European Conference on Model-Driven Architecture Foundations and Applications: Proceedings of the Tools and Consultancy Track

Centre for Telematics and Information Technology

Fifth European Conference on Model-Driven Architecture Foundations and Applications: Proceedings of the Tools and Consultancy Track

CTIT Workshop Proceedings

June, 2009
The Netherlands

Editor:
Regis Vogel

Regis Vogel (Eds.)

Fifth European Conference on Model-Driven Architecture Foundations and Applications: Proceedings of the Tools and Consultancy Track

Fifth European Conference on Model-Driven Architecture
Foundations and Applications (ECMDA-FA 2009)
University of Twente
Enschede, The Netherlands.
June 23 - 26 2009

Enschede, the Netherlands, 2009
CTIT Proceedings Series WP09-12
ISSN 0929-0672

Preface

This volume contains the proceedings of the Tools and Consultancy Track of the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2009). This is the written version of the presentations that will be held during the three sessions of the track. Presentations/slides are generally better understood when presented by a speaker. By compiling this document we provide the presenters a way communicate those details that are sometimes missing on the slide only material.
Like other technologies, to be successful MDA needs to have the right tools to support it. Its success is also dependant on practitioners with the knowledge to deploy it, and community knowledge to support its infusion. The role of the “Tools and Consultancy” track is to provide a time and space to present these aspects of MDA work. .
ECMDA-FA has traditionally had a very strong Tools and Consultancy track since this is often the meeting place between academic and industrial people interested in the practice of MDA. Attracting strong sponsors and academic and open source presenters, the 2009 Edition is representative of this strength.
The progress and evolution of MDA tools since the first edition in 2005 is notable. Similar to other technologies, MDA tools are maturing and provide more and more functionality. At the same time we have noticed an increase in the number of industrial (sponsors) participating in the track.
We would like to take this opportunity to thank the people who have contributed to the 2009 Tools and Consultancy track. We wish to thank all authors and reviewers for their valuable contributions, and we wish them a successful continuation of their work in this area. Finally, we want to thank the organization of the ECMDA-FA 2009 conference for the work and dedication that make this event possible.

June 2009

Regis Vogel

Organisation

Chair

Regis Vogel	The Information Highway Group

Local organizer

Arend Rensik	Univeristy of Twente
Ivan Kurtev	Univeristy of Twente
Andrey Sadovykh,	Softeam

Programme Committee

Regis Vogel, The Information Highway Group
Ivan Kurtev, Univeristy of Twente

Supporting Organisations

Modelplex
Centre of Telematics and Information Technology, University of Twente (CTIT)
Object Management Group (OMG)
Instituut voor Programmatuurkunde en Algoritmiek (IPA)
European Association for the Study of Science and Technology (EASST)

Sponsors

IBM-Rational software
Blu Age Software
Novulo
Sapiens
IBM Research
ModelioSoft
Pure-system
Shape

Table of Contents

Papyrus UML: an open source toolset for MDA.

Agnes Lanusse, Yann Tanguy, Huascar Espinoza, Chokri Mraidha, Sebastien Gerard
Patrick Tessier, Remi Schnekenburger, Hubert Dubois, François Terrier

CEA, LIST, Laboratoire d'ingénierie dirigée par les modèles pour les systèmes embarqués (LISE), Boîte courrier 94, GIF SUR YVETTE CEDEX, F-91191 France
{firstname.lastname@cea.fr}

Abstract. MDA is with no doubt a very good paradigm to support company teams all over the design and development process of software systems. This approach can be even more effective if tooling support is close to process practices and concepts used in application domains. This was one of the main motivations at the origin of Papyrus UML. This toolset is a general purpose UML 2 graphical modeller (http://www.papyrusuml.org/). Its main strength besides its strong compliance to UML 2 relies on its ability to exploit all the expressive power of advanced profiles management including static profiles to achieve cutting edge customization leading not only to profile storage and application but really to tool customization for domain specific applications. In the presentation, we illustrate this feature of the tool on a customization example to support EAST-ADL, an architecture description language for automotive. In a second part, we show how MARTE profile can be used to annotate models with temporal constraints and perform schedulability analysis at an early stage of modelling. Finally, we present code generation facilities.

Introduction

MDA has widely been recognized as a major advance in the design and development of complex systems, mainly thanks to separation of concerns promoted by the approach. Its benefit is expected to be even greater for real-time embedded systems that must respond to rapidly evolving target platforms and yet increasing functional complexity. Providing efficient tooling to support MDA designers for real-time embedded systems in their tasks is becoming the next challenge. With the emergence of large collaborative environments such as Eclipse projects, it becomes possible to capitalize on technologies and provide tools responding to large consensus expressed in OMG standards. UML 2 and its extension mechanisms based on profiles and stereotypes provide a way to maximize investment made on tools, since customization toward specific domains can be implemented through profile application and related tooling facilities as long as the tool platform exploits fully these concepts. It is the direction we have chosen and which is exposed here through a customization example to support EAST-ADL 2 automotive language (http://www.atesst.org/).

Papyrus has also been used for component based development for embedded systems using the eC3M (embedded Component Container Connector Model) environment (http://www.ec3m.net/) in telecommunication domain.

Papyrus, an open source UML 2 toolset

Started as a CEA LIST open source project, Papyrus is first of all a general purpose UML2 graphical modeller based on the Eclipse environment. Because modelling is not enough for fully claiming to be an MDE tool, it provides also code generation (C, C++, java) and facilitates external tools connection (schedulability analysis) in order to enable models to be the driving artefacts of the development process.

But Papyrus is also a very powerful tool for designing Domain Specific Modelling Languages (DSMLs) using the UML profile concept, and comes with a set of pre-defined extension plug-ins (UML 2 profiles) devoted to real-time embedded applications, including SysML, MARTE, CCM and LwCCM OMG standards. These profiles are packaged in plug-ins easily installable from Papyrus website feature (http://www.papyrusuml.org/) via Eclipse remote installation.

Papyrus leverages various Eclipse software components (plug-ins) to provide an efficient graphical editor for UML 2. Indeed, UML 2 specification compliance is one of the key objectives guiding Papyrus development. Hence, Papyrus currently supports eight of the diagrams described in the specification: **class** diagram, **component** diagram, **activity** diagram, **composite structure** diagram, **state machine** diagram, **use case** diagram, **sequence** diagram and **deployment** diagram.

Validation facilities are offered in order to reduce modelling errors, and or keep conformance to company or specific standard recommendations. These facilities rely on the extensible Eclipse EMFT Validation framework which is integrated in Papyrus. Users can rely on EMFT Validation extension points to add dedicated rules either externally to promote company or language specific modelling or within models. OCL or Java rules can be used for that purpose. Such rules (e.g. OCL constraints) benefit from text completion features implemented in Papyrus. They may be checked via an OCL verification engine, and detected errors can be located in the editor.

Advanced customization facilities

Thanks to the Eclipse framework, Papyrus is highly customizable using most of existing Eclipse extension points (e.g. menus, contextual actions, views, model transformations). In Papyrus the customizations are most of the time closely linked to profiles. Profiling is a key feature of UML that allows user to **adapt** the language to **specific modelling aspects** or **to some business domain and/or process**. Papyrus offers a graphical editor to create new profiles, and define subsequent new modelling concepts. These additional modelling concepts (called stereotypes) may be associated to specific modelling rules (in OCL) and/or specific graphical notations. Moreover, the concept of static profile is used to customize the toolset itself (e.g. iconization of concepts, palette customization), but also for implementing validation rules.

Indeed, as profiles enhance the UML2 language, the tool also needs to be extended to support the language extensions. The main idea behind this is: partially or completely mask UML elements if needed and preferably offer the possibility to use language specific concepts in the model. To achieve this, palette creation tools can be defined and used in Papyrus for a One-click creation of stereotyped elements. Such palette is visible on right side of Fig.1 which gives an overview of the tool customizations developed for the automotive EAST-ADL2.

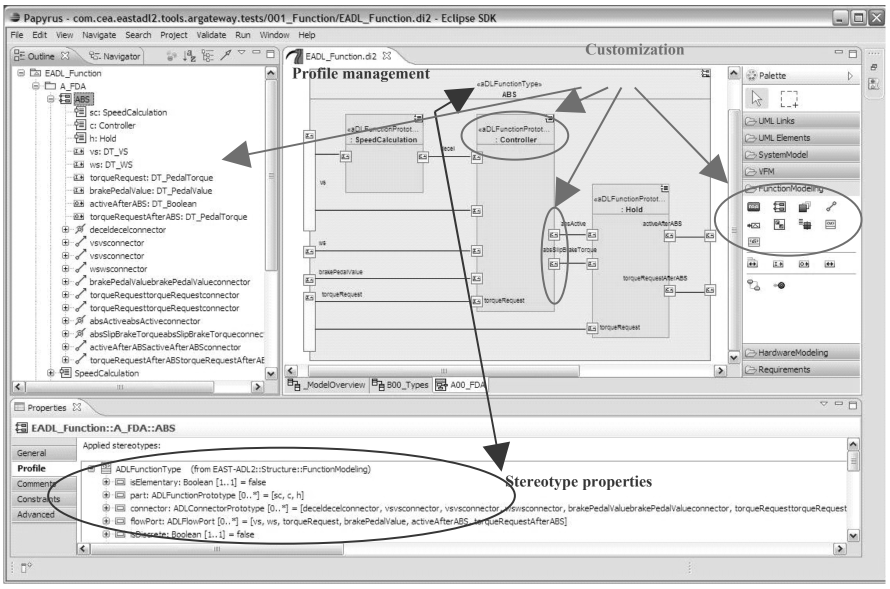


Fig. 1. Papyrus customization for the EAST-ADL2 language.

The graphical representation of model elements is also customizable with a set of replacement icons related to the language that are more appropriate than UML default icons. Again, Fig.1 gives a good example showing icons associated to automotive concepts instead of UML Class or Properties in diagrams and in the outline as well. Moreover, the graphical customization can be associated not only to profile stereotypes but with variations depending on the usage context. As an example, a stereotyped UML Property can have distinct representations depending on its type, or more generally depending on the value of any element defined in the model.

Embedded and Real-time MDA with UML and MARTE

The MARTE profile (http://www.omg.marte.org/) has been officially accepted by the Object Management Group (OMG) as an international standard in June, 2007, and a version 1.0 is expected to be available by June 2009. MARTE deals with time and resource constrained aspects, and includes a detailed taxonomy of hardware and software patterns along with their non-functional attributes to enable state-of-the-art quantitative analyses (e.g., performance and power consumption).

A key challenge of MARTE is to offer the tool support for making user models productive within a consistent development process. The tool demonstration will show some current results to integrate a set of tools enabling practitioners for incremental code generation, prototype execution, and timing analysis. In this approach, MARTE-annotated models, supporting a particular model of computation, are executed in a real-time framework and iteratively tuned (e.g., task allocation) by scheduling analysis tool results (see Fig.2). Once this is done, code generation can be triggered.

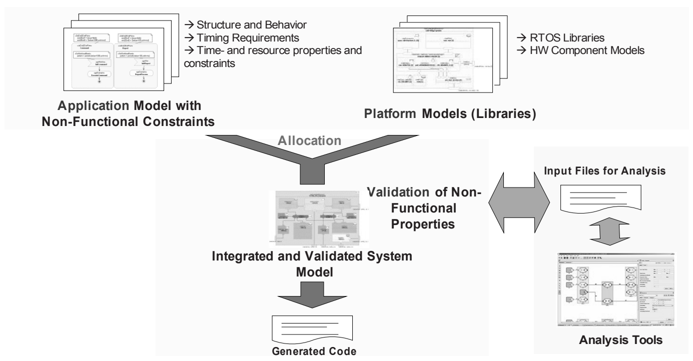

→ Structure and Behavior
→ Timing Requirements
→ Time- and resource properties and constraints
Application Model with Non-Functional Constraints
→ RTOS Libraries
→ HW Component Models
Platform Models (Libraries)
Allocation
Validation of Non-Functional Properties
Input Files for Analysis
Integrated and Validated System Model
Generated Code
Analysis Tools

Fig. 2. Some typical scenarios for the usage of the MARTE tool suite

MARTE enables scheduling analysis by providing standard *annotations*, which are used to add supplementary information to various kinds of UML elements that can then be interpreted by specialized tools or domain experts. For instance, fine-grained timing analyzers can help to determine the worst case execution times of relevant pieces of code, which are then used in scheduling analysis to predict end-to-end response times.

These annotations are defined in the Scheduling Analysis Modelling (SAM) subprofile of MARTE. The purpose is that these annotations can be applied to make a model look like a scheduling analysis model. These can then be used by an automated scheduling analysis tool to determine the fundamental timing properties of a software design. We have implemented an Eclipse/Papyrus plug-in to extract system models annotated with SAM stereotypes and transform them into input files for the SymTA/S analysis tool, as well as the other way around to feed back analysis results. It is based on a model-to-model transformation coded in ATL. The demonstration will detail a typical scenario for the usage of MARTE in this context.

Papyrus, now an Eclipse MDT project for UML and SysML

A new Eclipse MDT project led by CEA LIST called Papyrus started in 2008 thanks to the convergence of three main European open source initiatives. Papyrus, Top-Cased UML and Moskitt development teams are unifying their efforts with the goal to make out of these separate UML & SysML modelling tools, a common offer of outstanding performance and user-friendliness. This proposal was accepted by the Eclipse consortium and the project Eclipse web page is accessible at this link: http://www.eclipse.org/modeling/mdt/. Contact: sebastien.gerard[at]cea.fr.

MOCAS: a Model-Based Approach for Building Self-Adaptive Software Components

Cyril Ballagny, Nabil Hameurlain, and Franck Barbier

[1] LIUPPA, University of Pau
[2] BP1155, 64013 Pau, France
{cyril.ballagny,nabil.hameurlain,franck.barbier}@univ-pau.fr

Abstract. This paper describes MOCAS (Model Of Components for Adaptive Systems), a state-based component model which enables the self-adaptation of software components (behavior, implementation, properties) together with their coordination. A MOCAS component has its structure constrained by a UML profile. It embeds a UML state machine to realize its behavior at runtime. MOCAS relies on three different tools: a plug-in of the Eclipse-based TopCased platform for modeling MOCAS components and generating Java code, a engine for executing UML state machines running on the top of the Eclipse modeling framework (EMF) and an administration platform for managing MOCAS components.

1 Introduction

As the number and complexity of software systems increase, their sole management by humans becomes strenuous. Thus, software systems need to be able to manage themselves to free human administrators from repetitive tasks. Such systems are qualified as *autonomic systems* [1] and are able to manage their behavior and their relashionships in relation with third-party systems. An autonomic system is first and foremost a self-adaptive system: it is able to modify its behavior at runtime. Self-adaptation capabilities are required because of evolutions of environmental conditions, failures or the need for incorporating new components into running applications.

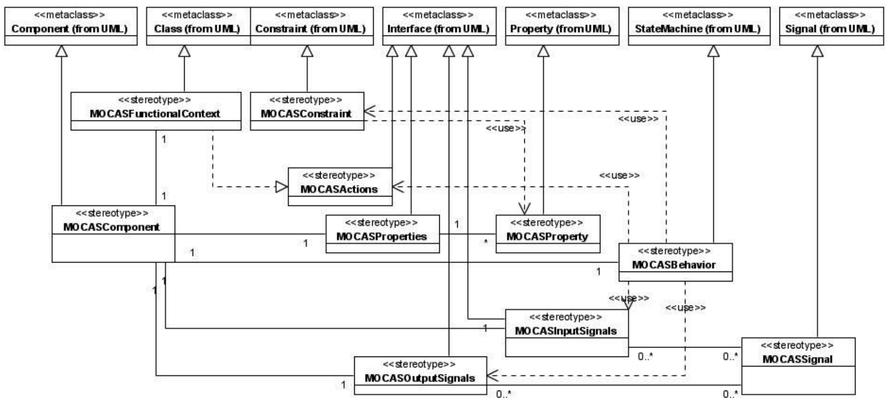
In this context, MOCAS is a state-based component model which enables the self-adaptation of software components (behavior, implementation, properties) together with their coordination [2]. MOCAS promotes usability and instruments self-adaptive systems based on models and model-driven engineering technologies. By usability, we consider the way to design self-adaptive systems, to develop them, to manage them and the way communication is carried out between different stakeholders (designers, developers, administrators,...). To that purpose, MOCAS has strong links with the Unified Modeling Language (UML).

After briefly exposing the MOCAS component model and its UML profile, we present three tools which have been realized to ease the design and development of a MOCAS system, to allow the execution of UML state machine models and to observe and control MOCAS components at runtime.

2 The MOCAS component model

A MOCAS *component* embeds at runtime a UML state machine model to describe and realize its *behavior*. The state machine controls the processing of *input signals* and triggers *internal actions* (like computation and data transformation) and *output of signals*. *Internal actions* are realized by the *functional context* of the component. The component supports *business properties* which are used by the *behavior*, to specify its *constraints* (such as transition guards and state invariants), and the *functional context*, to perform the business of the component. MOCAS components communicate *asynchronously* through signals. *Signals* allow MOCAS components to be loosely coupled and can inherit from other signals. These different concepts related to MOCAS are expressed as stereotypes and are contained in the UML profile in Fig. 1.

Each input signal is processed according to the *run-to-completion* cycle of the state machine. When a signal incomes, it is put at the end of the component message queue waiting for to be processed. Only one signal is processed at a time. A run-to-completion cycle starts by picking up the first signal of the queue and ends when all the transitions of the state machine have been fired and a new state configuration is active. Then a new cycle occurs by processing the next signal from the queue.



<<metaclass>> Component (from UML)
<<metaclass>> Class (from UML)
<<metaclass>> Constraint (from UML)
<<metaclass>> Interface (from UML)
<<metaclass>> Property (from UML)
<<metaclass>> StateMachine (from UML)
<<metaclass>> Signal (from UML)
<<stereotype>> MOCASFunctionalContext
<<stereotype>> MOCASConstraint
<<stereotype>> MOCASActions
<<stereotype>> MOCASComponent
<<stereotype>> MOCASProperties
<<stereotype>> MOCASProperty
<<stereotype>> MOCASBehavior
<<stereotype>> MOCASInputSignals
<<stereotype>> MOCASOutputSignals
<<stereotype>> MOCASSignal
<<use>>
1
*
0..*

Fig. 1. The MOCAS component UML profile

A MOCAS component has the distinctive feature to support its self-adaptation. To become *adaptive*, a MOCAS component is wrapped in a container conforming to the MOCAS component model. The container is in charge of managing the adaptation process: it checks the conformity of the requested adaptation according to the current component behavior, puts the component in a quiescient state [3], performs the adaptation operations and ensures the consistency of the whole process. To become *autonomic* so that it can perform the adaptation itself, a

MOCAS component is endowed with a control loop. The control loop in MOCAS is a set of MOCAS components: *sensors* detect what happens in the autonomic component environment and supervise its behavior, an *aggregator* centralizes and reports the sensed information to other components, an *evaluator* decides the actions to perform according to a policy and the reported information, and *effectors* realize adaptation by means of the decided actions. These different concepts related to the self-adaptation are expressed as stereotypes and are contained in the UML profile in Fig. 2.

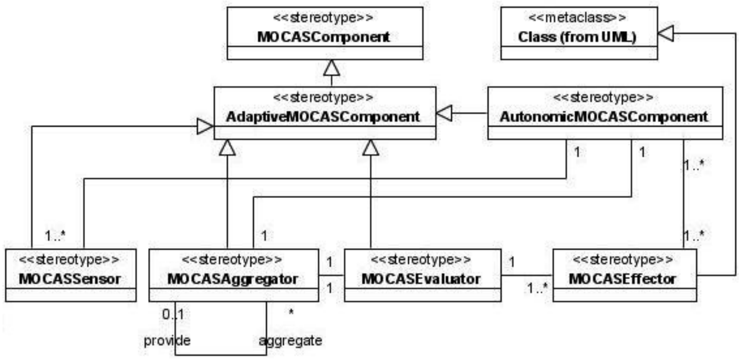


Fig. 2. The autonomic MOCAS component UML profile

3 A tool for development

As they rely on state machine models and UML profiles, MOCAS components are meant to be developed in CASE (Computer-Aided Software Engineering) tools supporting MDE. To that purpose, we choose to develop a plug-in for the Eclipse-based TopCased platform [4]. TopCased is an open-source project chiefly dedicated to the conception of embedded systems for aeronautics by putting forward the use of models. Eclipse already has a strong support for models thanks to their modeling framework (EMF) and their implementation of the UML metamodel. TopCased adds to it a lot of plug-ins supporting specification of OCL constraints [5], code generation, model verification and validation. Thereby, a TopCased plug-in has been easily developed by graduate students. This plug-in allows the specification of MOCAS components with the UML profiles introduced above. After verifying and validating models, it enables generation of Java code corresponding to the structure of each component. Finally, it creates a Jar file embedding the components and their state machine models.

4 A tool for execution

A MOCAS component embeds at runtime the state machine model describing its behavior. The state machine is loaded from the xmi file [6] generated by TopCased and is dynamically executed thanks to the MOCAS engine[3]. The MOCAS engine is a Java library which supports several UML state machine features such as orthogonal, submachine, history and final states, completion transitions, call operation and send signal actions, hierarchy of signals, guards and invariants,...

The first idea when designing the MOCAS engine was to rely on the JavaBeans component model, JavaBeans properties and Java reflexive capabilities. Thus, the elements used by the MOCAS engine are Java classes with a no-parameter public constructor and getter/setter to access properties. All the MOCAS stereotypes are realized by a class whose name is the same than the stereotype and which extends the EMF class matching the UML metamodel elements extended by the stereotype. As an exemple, the *MOCASSignal* stereotype is realized as shown in table 1. Then, all the signals processed by the state machine engine have to extend the

Table 1. Realization of the MOCASSignal stereotype in the MOCAS engine

```
import org.eclipse.uml2.uml.internal.impl.SignalImpl;
public class MOCASSignal extends SignalImpl{
        ...
}
```

MOCASSignal class. Moreover, the MOCAS engine also exists for the Java mobile platform thanks to a partial J2ME implementation of the UML metamodel[4]. As reflexive mechanisms are not available in this mobile environment, each MOCAS class has an attached class describing the methods which can be invoked.

5 A tool for administration

Each MOCAS component can be managed thanks to the MOCAS administration platform (MOCASAP). MOCASAP allows to deploy MOCAS components, which are contained in the Jar files generated by the plug-in, on a local host. It also allows to control their state machines (cf. in Fig. 4), to link MOCAS components dynamically, to deliver them updates, to switch their behavior and to send signals to components. Especially, MOCASAP enables to control the behavior of a MOCAS component by activating a specific state of its state machine. This is done by double clicking on a state and is useful in case of a failure, when it is required to rollback to a previous state known as stable.

[3] http://mocasengine.sourceforge.net/
[4] http://uml2forjava.sourceforge.net/

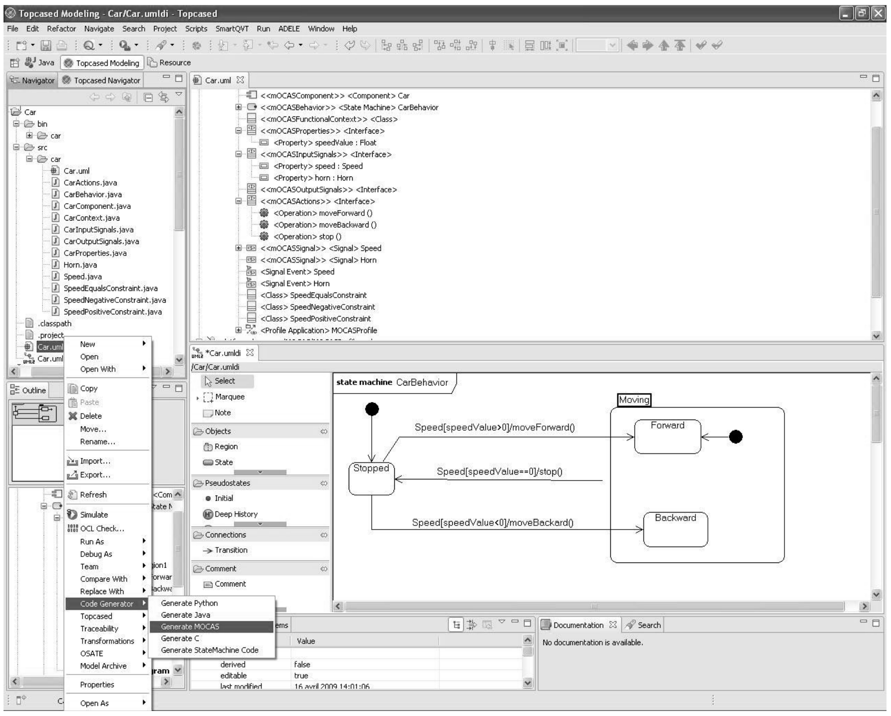
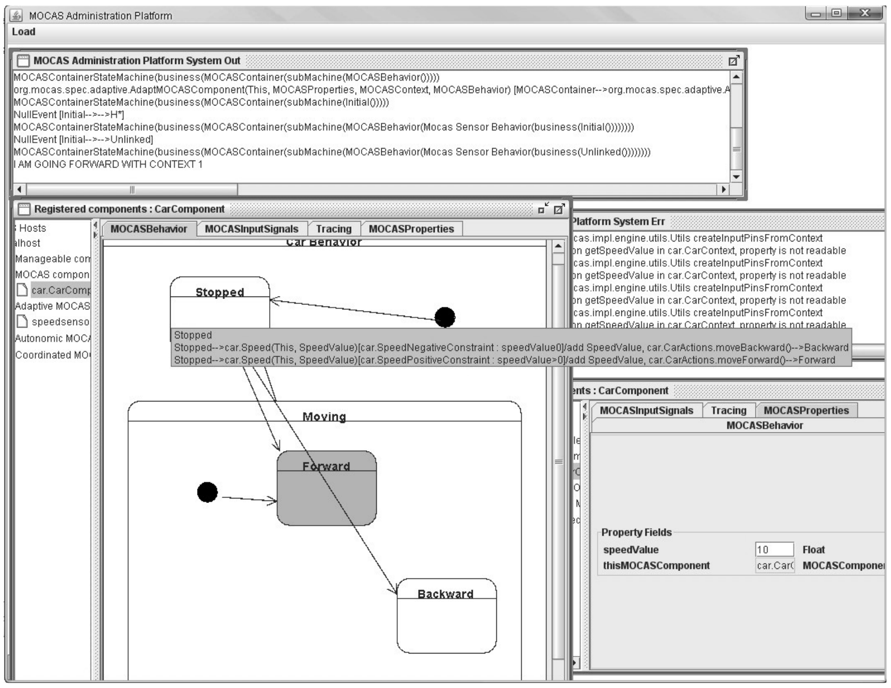

Topcased Modeling - Car/Car.umldi - Topcased
<<mOCASComponent>> <Component> Car
<<mOCASBehavior>> <State Machine> CarBehavior
<<mOCASFunctionalContext>> <Class>
<<mOCASProperties>> <Interface>
<Property> speedValue : Float
<<mOCASInputSignals>> <Interface>
<Property> speed : Speed
<Property> horn : Horn
<<mOCASOutputSignals>> <Interface>
<<mOCASActions>> <Interface>
<Operation> moveForward ()
<Operation> moveBackward ()
<Operation> stop ()
<<mOCASSignal>> <Signal> Speed
<<mOCASSignal>> <Signal> Horn
<Signal Event> Speed
<Signal Event> Horn
<Class> SpeedEqualsConstraint
<Class> SpeedNegativeConstraint
<Class> SpeedPositiveConstraint
<Profile Application> MOCASProfile
state machine CarBehavior
Moving
Forward
Backward
Stopped
Speed[speedValue>0]/moveForward()
Speed[speedValue==0]/stop()
Speed[speedValue<0]/moveBackard()
Code Generator
Generate Python
Generate Java
Generate MOCAS
Generate C
Generate StateMachine Code
No documentation is available.

Fig. 3. Specification of a MOCAS component in TopCased



```
MOCASContainerStateMachine(business(MOCASContainer(subMachine(MOCASBehavior()))))
org.mocas.spec.adaptive.AdaptMOCASComponent(This, MOCASProperties, MOCASContext, MOCASBehavior) [MOCASContainer-->org.mocas.spec.adaptive.A
MOCASContainerStateMachine(business(MOCASContainer(subMachine(Initial()))))
NullEvent [Initial-->-->H*]
MOCASContainerStateMachine(business(MOCASContainer(subMachine(MOCASBehavior(Mocas Sensor Behavior(business(Initial())))))))
NullEvent [Initial-->-->Unlinked]
MOCASContainerStateMachine(business(MOCASContainer(subMachine(MOCASBehavior(Mocas Sensor Behavior(business(Unlinked())))))))
I AM GOING FORWARD WITH CONTEXT 1
```

```
cas.impl.engine.utils.Utils createInputPinsFromContext
on getSpeedValue in car.CarContext, property is not readable
cas.impl.engine.utils.Utils createInputPinsFromContext
on getSpeedValue in car.CarContext, property is not readable
cas.impl.engine.utils.Utils createInputPinsFromContext
on getSpeedValue in car.CarContext, property is not readable
cas.impl.engine.utils.Utils createInputPinsFromContext
on getSpeedValue in car.CarContext, property is not readable
```

```
Stopped
Stopped-->car.Speed(This, SpeedValue)[car.SpeedNegativeConstraint : speedValue0]/add SpeedValue, car.CarActions.moveBackward()-->Backward
Stopped-->car.Speed(This, SpeedValue)[car.SpeedPositiveConstraint : speedValue>0]/add SpeedValue, car.CarActions.moveForward()-->Forward
```

speedValue	10	Float
thisMOCASComponent	car.Car(	MOCASComponen

Fig. 4. The MOCAS administration platform

6 Conclusion and future directions

MOCAS is a component model enabling self-adaptation of software components. MOCAS components are developed through a model-driven process. A MOCAS component is designed in the Eclipse-based TopCased platform by using UML. A UML profile constraints its structure and a state machine model describes its behavior. A MOCAS component embeds its state machine model at runtime. The state machine is executed thanks to the MOCAS engine. A MOCAS component is monitored and controlled thanks to the MOCAS administration platform. In the next tool releases, we want the TopCased plug-in to enable the specification of an assembly of MOCAS components. We will also extend the MOCAS engine features with time events and change of property values for triggering transitions. Finally, we are investigating the support of remote deployment and mobility of MOCAS components in the MOCAS administration platform.

References

1. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (2003) 41–50
2. Ballagny, C., Hameurlain, N., Barbier, F.: Dynamic adaptive software components: the MOCAS approach. In: ASBS'08: Proceedings of The First IEEE International Workshop on Autonomous and Autonomic Software-Based Systems, Cergy-Pontoise, France, ACM (2008) 517–524
3. Kramer, J., Magee, J.: The evolving philosophers problem: Dynamic change management. IEEE Transactions on Software Engineering **16**(11) (1990) 1293–1306
4. Farail, P., Gaufillet, P., Canals, A., Camus, C.L., Sciamma, D., Michel, P., Cregut, X., Pantel, M.: The TOPCASED project : a toolkit in open source for critical aeronautic systems design. In: ERTS2006: 3rd Embedded Real Time Software Conference, ACM (2006) 54–59
5. Object Management Group, Inc.: UML Object Constraint Language (OCL) 2.0 Specification. (May 2006)
6. Object Management Group, Inc.: XML Metadata Interchange Specification. (May 2003)

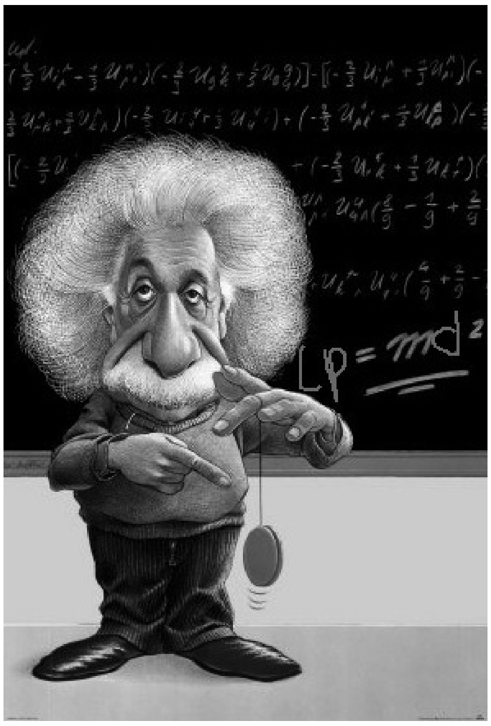
Modeling Your Future Success

Sagi Schlisser

Sapiens International, P.O. Box 4011, Nes Ziona 74140, Israel

Abstract. MDD is supposed to revolutionize the development of software through models which are, as far as possible, substituted for code. Code is nothing else than an operational model that includes all of the required details, which themselves relate to runtime platforms. But, solving technical problems is not enough. Sapiens eMerge EMDD goes beyond MDD by enabling a true codeless environment that provides an enterprise-ready container. EMDM handles modeling while promoting agility by incorporating concrete material in the early phase of development—like GUIs and business rules. EMDD promotes the consistent integration of models, rules, and GUIs. It also emphasizes engineering techniques and model inspection that really take into account the seven sins of modeling.

Keywords: modeling, model-driven development, MDD, business rules

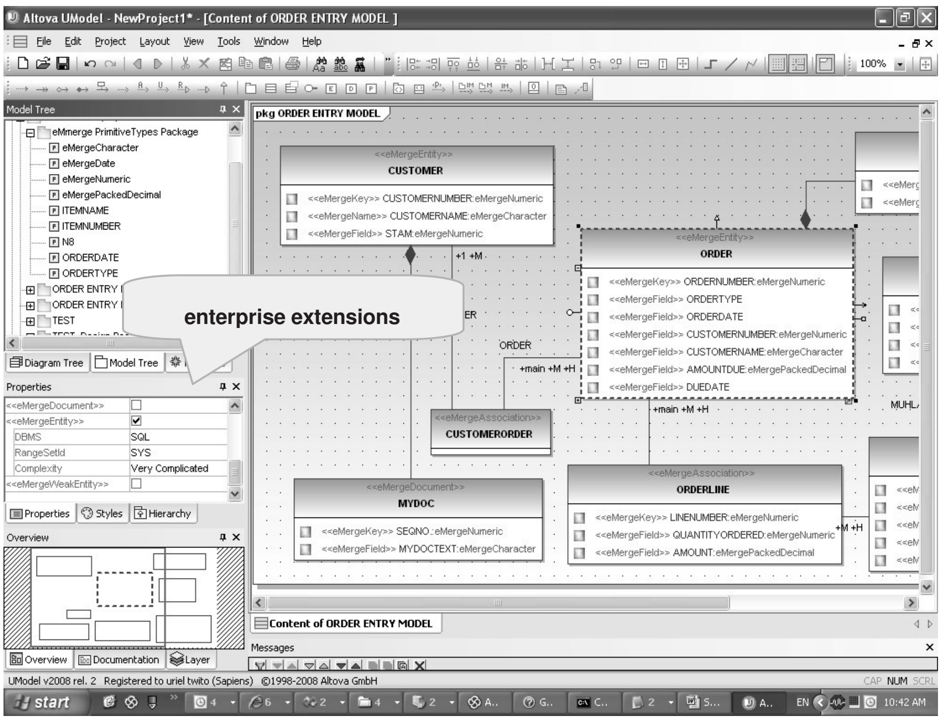


1. Model-Driven Development

In the late ‘90’s the maturing of object-oriented (OO) modeling lead to the unifying OO methods of the Unified Modeling Language UML®. Even though there was significant progress, software production remained small-scale and the need for developing modern, heavy-duty approaches became obvious. In the early 2000’s, the Model Driven Architecture® (MDA®) initiative tried to tackle this problem by laying down the foundations of model-centric development. Beyond the complex technological nature of MD*x* (*x* = Architecture, Engineering or Development), potential users—who intend to investigate this software development paradigm—obviously expect significant economical progress, because:

- Modeling leads to an increase in productivity and a faster time-to-market.
- MDx’s roots in object orientation leads to reusability and maintainability.

However, MDA is disadvantaged by the complex nature of modeling and by models that are neither accessible nor understandable. Models—due to their abstract nature—cannot be qualified as natural nor as intuitive. Software developers prefer tangible software artifacts, like GUIs, in place of models. (as formulated by B. Meyer)


Altova UModel - NewProject1* - [Content of ORDER ENTRY MODEL]
enterprise extensions
pkg ORDER ENTRY MODEL
<<eMergeEntity>> CUSTOMER
<<eMergeKey>> CUSTOMERNUMBER:eMergeNumeric
<<eMergeName>> CUSTOMERNAME:eMergeCharacter
<<eMergeField>> STAM:eMergeNumeric
<<eMergeEntity>> ORDER
<<eMergeKey>> ORDERNUMBER:eMergeNumeric
<<eMergeField>> ORDERTYPE
<<eMergeField>> ORDERDATE
<<eMergeField>> CUSTOMERNUMBER:eMergeNumeric
<<eMergeField>> CUSTOMERNAME:eMergeCharacter
<<eMergeField>> AMOUNTDUE:eMergePackedDecimal
<<eMergeField>> DUEDATE
<<eMergeAssociation>> CUSTOMERORDER
<<eMergeDocument>> MYDOC
<<eMergeKey>> SEQNO.:eMergeNumeric
<<eMergeField>> MYDOCTEXT:eMergeCharacter
<<eMergeAssociation>> ORDERLINE
<<eMergeKey>> LINENUMBER:eMergeNumeric
<<eMergeField>> QUANTITYORDERED:eMergeNumeric
<<eMergeField>> AMOUNT:eMergePackedDecimal
eMmerge PrimitiveTypes Package
eMergeCharacter
eMergeDate
eMergeNumeric
eMergePackedDecimal
ITEMNAME
ITEMNUMBER
N8
ORDERDATE
ORDERTYPE
Model Tree
Properties
Overview

<<eMergeDocument>>	☐
<<eMergeEntity>>	☑
DBMS	SQL
RangeSetId	SYS
Complexity	Very Complicated
<<eMergeWeakEntity>>	☐

UModel v2008 rel. 2 Registered to uriel twito (Sapiens) ©1998-2008 Altova GmbH

Fig.1: Sapiens eMerge XMI model with enterprise UML extensions (shown with Altova Umodel)

2. Blueprints for Enterprise Model-Driven Development

Producing a new mission-critical enterprise application is no small feat, especially when dealing with high-end, complex applications. These applications must adhere to enterprise standards, such as:

- Monitoring
- Fault tolerance
- Journaling and audit trail
- High performance
- Scalability
- Clustering
- Security

Achieving a mission-critical level must be done economically—both from the software and from the deployment point of view—in order to be able to service large numbers of concurrent users (back-office, partners, web, etc.). From a development point of view, analyzing many (or even most) mission-critical level applications, one finds the challenges for most applications are split between:

- Object modeling: Modeling the objects in the application and the relationships between them.
- Persist objects: Managing the object relational relationship to persist the data in a standard RDBS.
- Business rules: Representing, collecting, and maintaining the business logic as business rules.
- Events: Managing the expected business events.
- Presentation: Creating the user interface.
- Integration: Integrating the above into one successful application.

The biggest challenge to successful enterprise application development and a large part of the driving force for modeling is the business—technical gap. That is, enabling the engineering team to talk and to communicate with the business team.

The MD*x* challenge is to bring all of the above challenges into a viable solution where the model is so close to the actual business that the model can be executable—that is *model executability*. But, merely generating code or executing models as is being done today (even the more advanced MD*x*'s using frameworks) do not meet the enterprise challenges described above.

The answer lies in Model-Driven Middleware (MDM): The ability to execute the model without generating code on model-aware middleware. Such middleware—together with support for extended modeling, presentation, and business rules—can and does run mission-critical enterprise applications with thousands of users and millions of transactions a day.

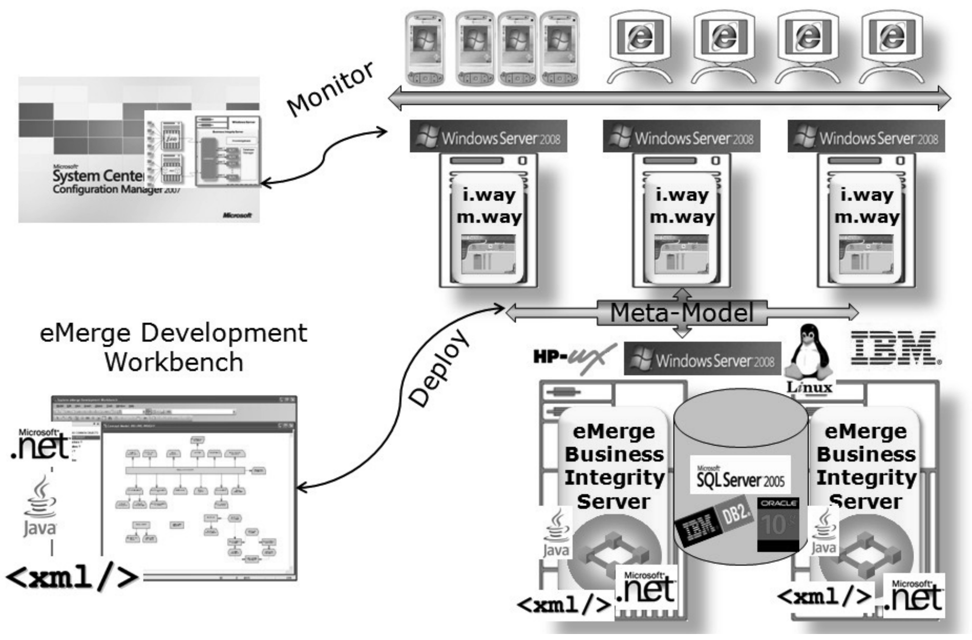
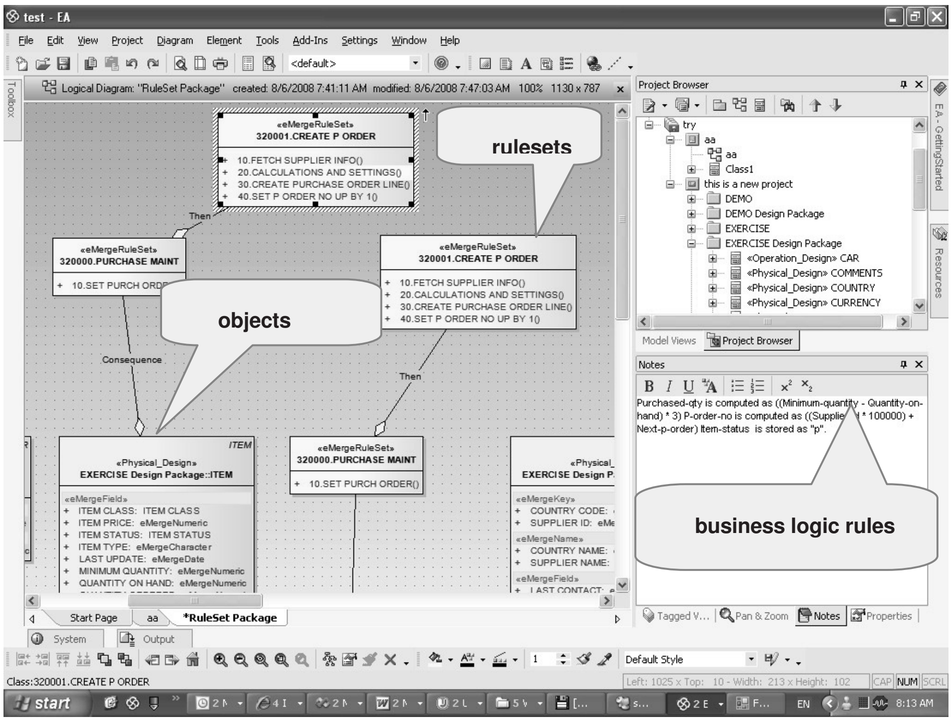


Fig.2: Sapiens eMerge Enterprise Model-Driven Middleware (EMDM) with Java and .NET integration

The Seven Sins of Modeling

The seven sins of modeling (as formulated by B. Meyer) are: ambiguity, contradiction, forward reference, noise, over specification, silence and wishful thinking. The obvious expected outcome of MDD is how to better address requirements engineering issues: the presumed advantage of models compared to code. Models, through their abstract nature, favor early detection of problems, these being omissions, requirement misunderstandings, and so on. There is no miracle. An MDM based on a metadata repository allows for active inspection and interrogation of the model, together with the business rules and the presentation. Model inspection and model reporting lead to productivity and enable sin-detection and analysis.


test - EA
rulesets
objects
business logic rules
«eMergeRuleSet» 320001.CREATE P ORDER
10.FETCH SUPPLIER INFO()
20.CALCULATIONS AND SETTINGS()
30.CREATE PURCHASE ORDER LINE()
40.SET P ORDER NO UP BY 1()
«eMergeRuleSet» 320000.PURCHASE MAINT
10.SET PURCH ORDER()
Then
Consequence
«Physical_Design» EXERCISE Design Package::ITEM
Project Browser
Notes

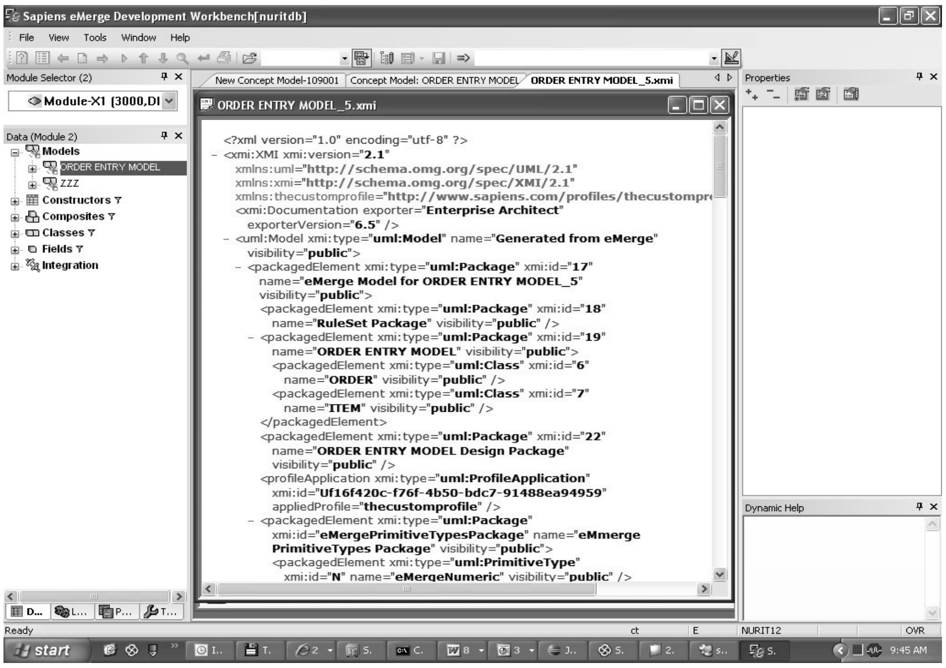
Purchased-qty is computed as ((Minimum-quantity - Quantity-on-hand) * 3) P-order-no is computed as ((Supplie d * 100000) + Next-p-order) Item-status is stored as "p".

Fig.3: Sapiens eMerge EMDM modeling transparency (exported to EA)

3. EMDM: The Model is the Code and The Code is the Model

MDD emphasizes metamodeling, XML-based model and metamodel formats, open extensible model transformation languages and meta-languages, and consequently, corresponding tools for managing models and their transformations. The main question is: *To what point is the delivery of decorated models with platform-dependent features realistic?* That is, do we really believe that we can do *it* without writing code? In practice, the MDD process currently used stops when the material in models do not enable the automatic generation of code. Therefore, developers have to provide additional implementation details and tuning. The difficulty is the control of such additions to models; the models are marketed as finished, but in many cases the additional details and tuning become sizeable.

Sapiens Enterprise Model-Driven Middleware (EMDM) is built around the concept: *The model is the code and the code is the model.* This powerful concept means the middleware is used to execute the model directly without generating any code. This is crucial for enterprise model executability, since mission-critical applications are exponentially more complex than smaller applications. Mission critical applications are where code generation breaks down and model executability stops.



```
  <?xml version="1.0" encoding="utf-8" ?>
- <xmi:XMI xmi:version="2.1"
    xmlns:uml="http://schema.omg.org/spec/UML/2.1"
    xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
    xmlns:thecustomprofile="http://www.sapiens.com/profiles/thecustompre
    <xmi:Documentation exporter="Enterprise Architect"
      exporterVersion="6.5" />
  - <uml:Model xmi:type="uml:Model" name="Generated from eMerge"
      visibility="public">
    - <packagedElement xmi:type="uml:Package" xmi:id="17"
        name="eMerge Model for ORDER ENTRY MODEL_5"
        visibility="public">
        <packagedElement xmi:type="uml:Package" xmi:id="18"
          name="RuleSet Package" visibility="public" />
      - <packagedElement xmi:type="uml:Package" xmi:id="19"
          name="ORDER ENTRY MODEL" visibility="public">
          <packagedElement xmi:type="uml:Class" xmi:id="6"
            name="ORDER" visibility="public" />
          <packagedElement xmi:type="uml:Class" xmi:id="7"
            name="ITEM" visibility="public" />
        </packagedElement>
        <packagedElement xmi:type="uml:Package" xmi:id="22"
          name="ORDER ENTRY MODEL Design Package"
          visibility="public" />
        <profileApplication xmi:type="uml:ProfileApplication"
          xmi:id="Uf16f420c-f76f-4b50-bdc7-91488ea94959"
          appliedProfile="thecustomprofile" />
      - <packagedElement xmi:type="uml:Package"
          xmi:id="eMergePrimitiveTypesPackage" name="eMmerge
          PrimitiveTypes Package" visibility="public">
          <packagedElement xmi:type="uml:PrimitiveType"
            xmi:id="N" name="eMergeNumeric" visibility="public" />
```

Fig.4: Sapiens eMerge EMDM XML-based metamodel

4. Incremental Maintenance

A key expectation of MDD is the ability to easily update models due to new or adjusted client requirements. Such maintenance must be based on a rapid lifecycle and done in a cost effective way. Heavyweight changes—for instance, database restructuring—generate high costs, unlike lightweight modifications (which represent around 90% of all maintenance).

Sapiens eMerge EMDM's agility—associated with modeling traceability and many visual wizards for presentation and rules—helps cut maintenance costs dramatically. Changes are much more maintainable when the application is running on middleware; this is due to middleware change support and refactoring which does not depend on external code.

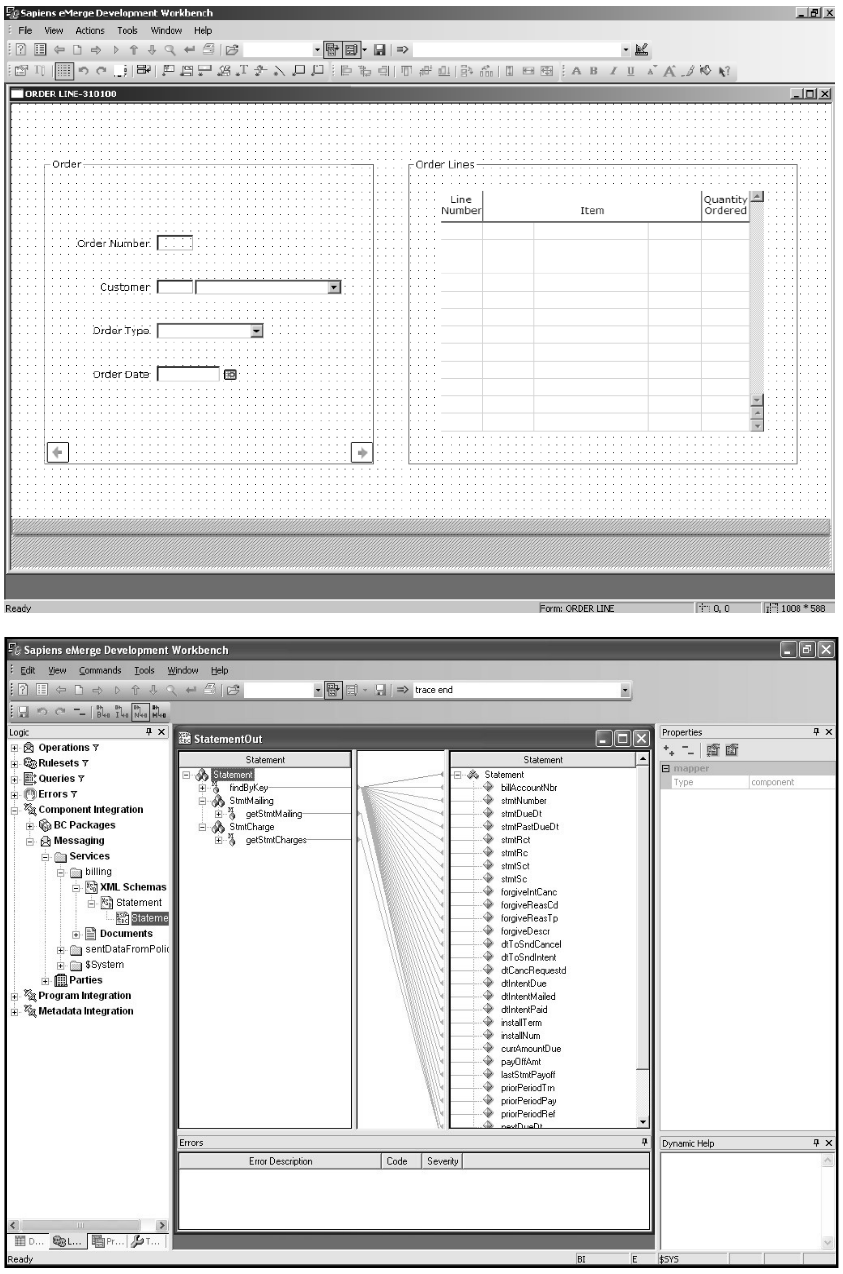


Fig.5: Sapiens eMerge EMDM agility—Rich Internet Application (RIA) builder and integration mapper

5. Conclusion

MDD is supposed to revolutionize the development of software through models which are, as far as possible, substituted for code. Code is nothing else than an operational model that includes all of the required details, which themselves relate to runtime platforms. But, solving technical problems is not enough. Engineers, developers, and end-users must be convinced that MDD is productive, cost-effective, simple, and intuitive. Sapiens eMerge EMDD goes beyond MDD by enabling a true codeless environment that provides an enterprise-ready container. EMDM handles modeling while promoting agility by incorporating concrete material in the early phase of development—like GUIs and business rules. EMDD promotes the consistent integration of models, rules, and GUIs. It also emphasizes engineering techniques and model inspection that really take into account the seven sins of modeling.

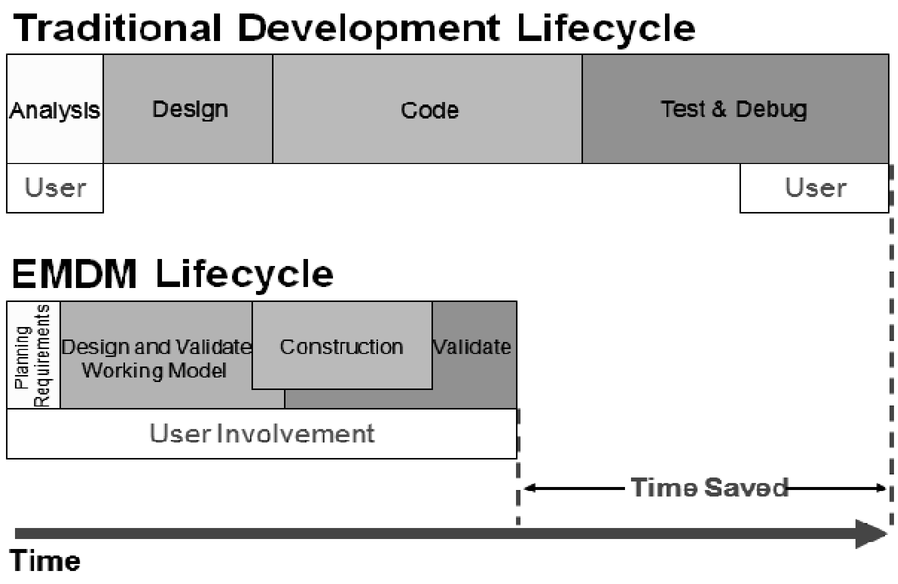


Fig.6: Sapiens EMDM – Modeling your future success

Modelio: Globalizing MDA

Philippe Desfray, SOFTEAM

Abstract. Modelio (www.modeliosoft.com) is a modeling tool addressing both Business and IT, that supports IS to business alignment. The MDA technology has been too much focused on software aspects, mainly on code generation or technical architecture aspects. MDA must be much more used to support methodologies and sustain modeling on the entire enterprise scope. This paper presents how, by integrating in a single repository the entire enterprise scope modeling coverage, by providing strong MDA capacities and by allowing an efficient and flexible distributed team cooperation, Modelio allows a global MDA approach, empowering thus MDA to the enterprise wide application.

Keywords: MDA, UML modeling, BPMN, Enterprise architecture, Goal, Dictionnary, Requirements, business rules modeling.

1 Introduction

Software development is only a small part of the problems to be managed within companies. Beyond software development issues are problems related to the definition of the company strategy, to the organization of the company, the definition of its business processes, and to the IS evolution linked to the existing IS.

The MDA technology has been too much focused on software aspects, mainly on code generation or technical architecture aspects. MDA must be much more used to support methodologies and sustain modeling on the entire enterprise scope.

The modeling scope must therefore be enterprise wide, and include scoping and guidance techniques such as goal analysis, dictionary and business rules analysis and requirements analysis.

Scaling up MDA implies large modeling capacities coverage, a strong MDA integration, and a strong distributed teamwork support. MDA is limited by the tool's capacities, to cover the enterprise wide modeling needs.

1 Modelio: a new range of tool for a complete coverage

Modelio provides a complete coverage to support the modeling activities for each stakeholder within a company. Its modeling support includes UML2, BPMN, Enterprise Architecture, SOA architectures, Goal analysis, requirement analysis, Dictionary and business rules analysis.

In addition to a predefined set of model driven code generation modules, Modelio provides a strong integrated MDA capacity, through its UML Profile editor, and its rich Java API for metamodel access, model transformation and tool customization.

Modelio supports distributed team work and model driven development cooperation through tight Subversion (SVN) integration. Flexible and efficient, the team cooperation with Modelio works with time to time internet access.

One repository for enterprise large modeling, integrated MDA, team cooperation support and integrated traceability management enables the MDA application globalization, from enterprise vision to development, and for large cooperating communities.

Modelio is distributed in three editions for different usages and budgets:

- Modelio Free: A comprehensive, professional modeling tool, free of charge!
- Modelio Java Express: Model-driven Java code generation for only €100!
- Modelio Enterprise: For projects and teams, extendable through a large set of modules and dedicated MDA extensions.

Modelio Enterprise Edition provides a large set of modules, that allow the user to configure it for specific usages.

- Modelio Modeler : UML2 modeling; Project sharing support
- Java Designer : Model driven Java code generation.
- C# Designer : Model driven C# code generation
- C++ Designer: Model driven C++ generation
- Document Publisher : Document generation (OpenXML-Word; HTML) . Document roundtrip edition (Word); Many on the shelf document templates
- Requirement Analyst: Model integrated requirement analysis; Impact analysis; Traceability management
- Dictionary & Business Rules: Model integrated Dictionary & Business rules analysis.
- Goal Analyst: Model integrated Goal analysis support.
- SVN Teamwork manager: Distributed teamwork. Configuration & version management. Subvertion integration.
- MDA Designer: UML profile editor, Java API for metamodel handling, model transformation, Modelio GUI customization

- SOA Architect: SOA architecture modelling
- EA – BPM Modeler: Enterprise Architecture and Business Process modelling. BPMN modelling.
- XML Designer: Generates or reverses XML schemas (XSD)
- WSDL Designer : Generates or reverses WSDL code from interfaces.

See demos, guidelines, download evaluate or buy Modelio on www.modeliosoft.com

pure::variants - Combining Variant Management and Model-Driven Development in Product Lines

Danilo Beuche, pure-systems, Germany

Abstract. In this demonstration the integration of pure::variants, a leading tool for variant management in (software) product lines (SPL), and model driven development (MDD) environments will be shown. The pure::variants framework currently integrates with tools suites for development of domain specific modeling languages (e.g. openArchitectureWare), generic modeling languages (UML e.g. Enterprise Architect) and domain specific modeling tools (e.g. MATLAB/Simulink).
The demonstration will briefly cover pure::variants basics such as feature modeling and will then show some use cases with SPL MDD approaches.

.

IBM Research work on MDE

Andrei Kirshin[1], Tali Yatzkar-Haham[1], Shiri Kremer-Davidson[1]

[1] IBM R&D Labs in Israel, Haifa University Campus, Mount Carmel,
Haifa, 31905, Israel
{kirshin, shiri, tali}@il.ibm.com

Abstract. This paper presents the work that is being done in IBM Research in the area of Model Driven Engineering (MDE). The talk contains three parts: Model-based Testing, Product Lines Engineering, and Telecom Service Creation Environment.

Model-based Testing: Model-based Testing already proved itself as a methodology that improves effectiveness and efficiency of testers. We developed a tool suite that leverages the power of UML and enables automatic test generation based on behavioral system models, debugging of such models and manual test editing.

Product Lines Engineering: Product Lines Engineering is rapidly emerging as an important paradigm, allowing order-of-magnitude improvements in time to market, maintenance cost, quality, and mass customization support. Our tools and methodologies facilitate software reuse, support integration of suppliers' components in the supply-chain domain, and enabling transformations into various product artifacts.

Telecom Service Creation Environment: Our work describes a model based approach for rapid creation of Next Generation Networks Services. Our tooling enables designers to create models of such services without knowledge of the underlying protocols and to generate runnable services from them without additional code being written.

Keywords: model-based testing, product lines, telecom service

1 Introduction

This paper presents the work that is being done in IBM Research in the area of Model Driven Engineering (MDE). Most of the tools presented here are developed in IBM Haifa Research Lab [1][2]. Extensions of model simulator were done in collaboration with IBM Tokyo Research Lab [3]. The paper does not cover all the work done by IBM Research division in this area. Section 2 presents Model-based Testing work, Section 3 talks about Product Lines Engineering, and Section 4 describes Telecom Service Creation Environment.

2 Model-based Testing

In Model Driven Engineering developers create models instead of writing the code. The code is automatically generated from the models.

Similarly, in model-based testing testers use models to describe the behavior of the system under test and the tests are generated automatically from these models. Fig. 2 briefly shows the model-based testing workflow.

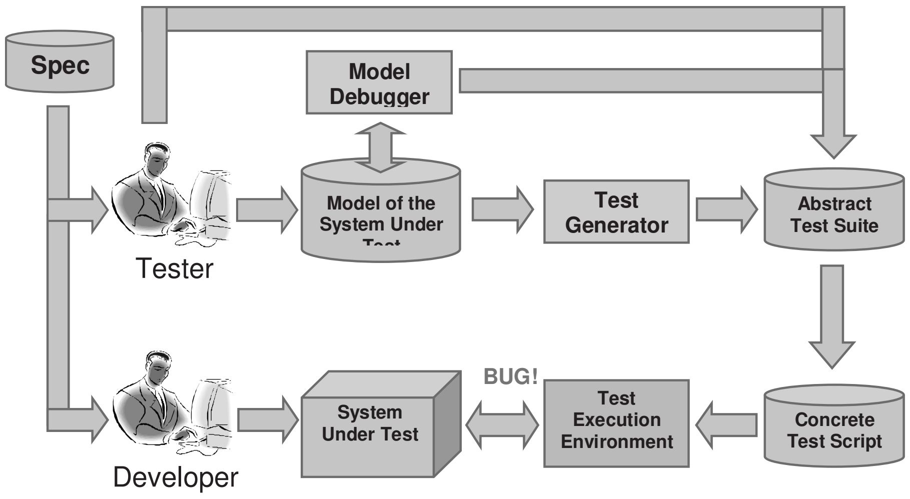


Fig. 1. Model-based testing

This section describes tools that were developed at the IBM Haifa Research Lab and used for model-based testing. They are extensions of Rational Software Architect (RSA), which is used to create UML models of the system under test. Typically, the structure of the system is described using class diagrams and composite structure diagrams. State machines, activities, and Java code snippets are used to describe the behavior. In our scenario, the Model Execution Engine [4] executes the model "behind the scenes", while the "main actors" are the model debugger and test generator (see Fig. 2).

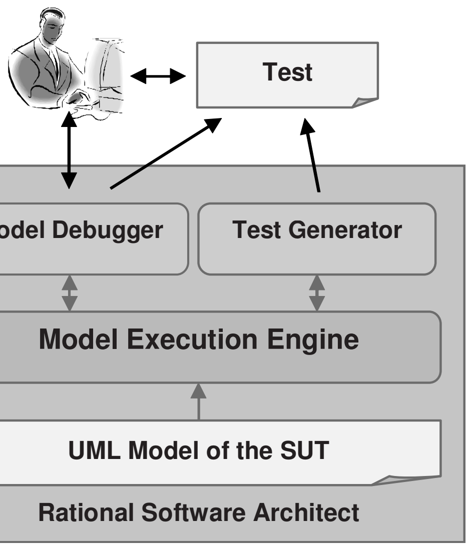


Fig. 2. The architecture of the tool suite

The model debugger verifies that the model describes the correct expected behavior of the System Under Test (SUT). It enables the user to interact with the executable model in two ways:

– *To control the execution* by sending input to the model; that is, create instances, invoke operations on instances, and send signals to instances.
– *To observe the execution* by observing the output of the model; that is, the attribute values, active states, and signals to the environment.

The model debugger helps answer the question "Is there a defect in the model?" at two different stages: before test generation during the modeling of the SUT, and after test generation when a test case fails. In the latter case, the defect can be in the SUT or in the model. If the defect is in the model (its behavior is wrong), the debugger localizes and fixes the defect, thus answering the question, "Where is the defect?" The debugger allows the setting of breakpoints on model elements. Fig. 3 illustrates a running state machine with a highlighted active state and running transition. It also shows breakpoints and execution pending elements.

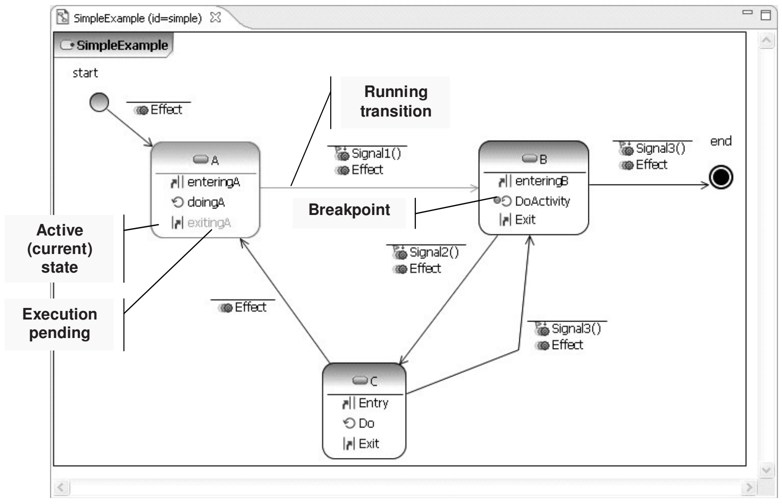


Fig. 3. A debugging session of a UML model

The test generator also uses the model execution engine and acts similarly to the model debugger by sending input and observing output. The input is recorded as test sequences of stimuli applied to the SUT. The output is also recorded as expected outcomes. In other words, the model is used as a test oracle predicting correct behavior. During test generation, the next input for a test sequence is selected by the test generator, depending on the coverage task chosen by the user:

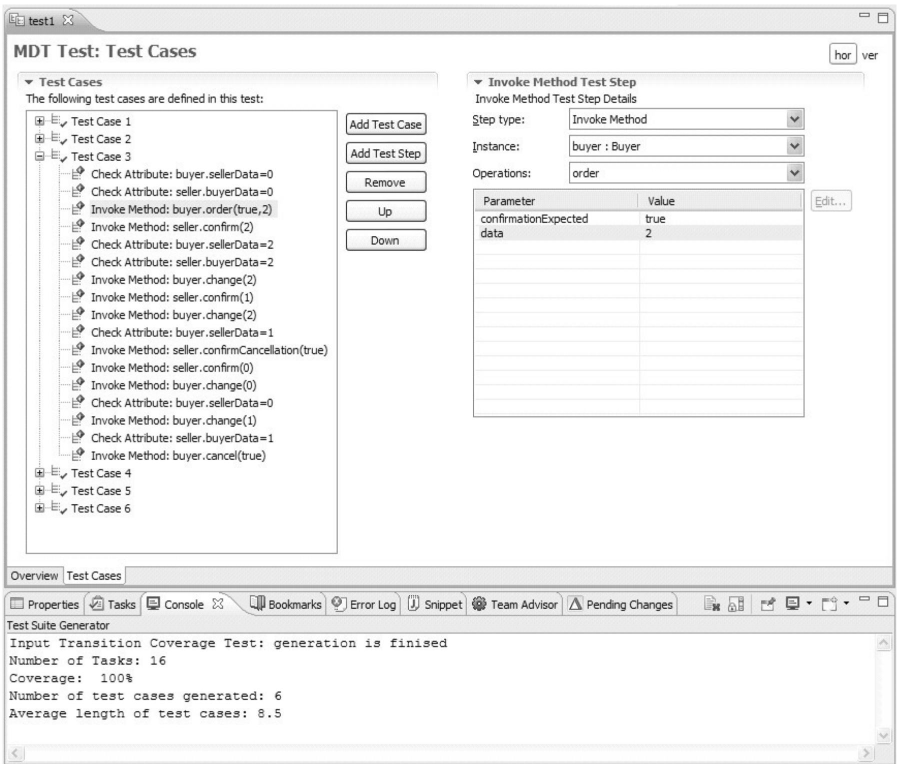
– *Random* generates random sequences of stimuli (no coverage)
– *Input Coverage* covers as much input as it can (a specific input)
– *Input Step Coverage* covers as much input in each step in the test case as it can (a specific input in a specific step in the test case)
– *Input Step Pair Coverage* covers as many different pairs of input in each pair of steps in the test case as it can (a specific pair of input values in a specific pair of steps).

The current version of the test generator only implements *input coverage algorithms*. Their advantage is that do not deal with the problem of state explosion. This problem is faced only when test generators explore the whole state space of the test model, which can grow very quickly for large test data.

Another advantage of the test generator is that it actually executes the model of the SUT. This enables us not only to generate the sequences of stimuli but also to precisely predict the expected behavior (output of the SUT). Test generators that statically analyze the model often have problems in predicting expected results.

Another strength of the test generator is that it uses the same model execution engine as the model debugger. If wrongly generated tests result from an incorrect test model, the problem can be easily located and fixed in the model using the model debugger. Test generators using their own execution engine might interpret the semantics of the model differently, which complicates the maintenance of the original model.

For the test suite format, we use the Eclipse Test & Performance Tools Platform (TPTP) [5]. Test steps (stimuli and observations) reference model elements and a special editor (see Fig. 4) developed at the IBM Haifa Research Lab is used for convenient viewing and editing of the generated tests. In addition to using the described test generator, tests can also be created manually using the editor. Finally, the test can be either translated to a specific test script for execution on the SUT, or alternatively, TPTP can be extended to execute the test directly on the SUT.



```
Test Case 1
Test Case 2
Test Case 3
    Check Attribute: buyer.sellerData=0
    Check Attribute: seller.buyerData=0
    Invoke Method: buyer.order(true,2)
    Invoke Method: seller.confirm(2)
    Check Attribute: buyer.sellerData=2
    Check Attribute: seller.buyerData=2
    Invoke Method: buyer.change(2)
    Invoke Method: seller.confirm(1)
    Invoke Method: buyer.change(2)
    Check Attribute: buyer.sellerData=1
    Invoke Method: seller.confirmCancellation(true)
    Invoke Method: seller.confirm(0)
    Invoke Method: buyer.change(0)
    Check Attribute: buyer.sellerData=0
    Invoke Method: buyer.change(1)
    Check Attribute: seller.buyerData=1
    Invoke Method: buyer.cancel(true)
Test Case 4
Test Case 5
Test Case 6
```

Step type: Invoke Method
Instance: buyer : Buyer
Operations: order

Parameter	Value
confirmationExpected	true
data	2

```
Input Transition Coverage Test: generation is finised
Number of Tasks: 16
Coverage:  100%
Number of test cases generated: 6
Average length of test cases: 8.5
```

Fig. 4. Test editor

This work was extended for the verification of embedded systems using collaborative simulation of SysML and Simulink models [6].

3 Model-Driven Product-Lines for Embedded Software and for Supply-Chain Companies

We present a model-driven development approach for the design and development of Software Product Lines. We suggest a model-driven component-based framework for software product lines development, together with a methodology for using the framework.

Software is becoming increasingly large and complex. As a result, companies want to reuse more of the software that they produce. Software product-lines engineering relates to engineering techniques for creating a portfolio of similar software systems from a shared set of software assets. While companies target the needs of their customers by creating a product-line, today's tools and techniques for system and software development tend to focus on individual products. Thus, developing software for a product-line using current tools becomes extremely complex.

Our software product line tools and methodologies facilitate software reuse, support integration, configuration and delivery of suppliers' components in the supply-chain domain, enabling transformations into various product artifacts, including build scripts and glue code for the integration of third party suppliers' components.

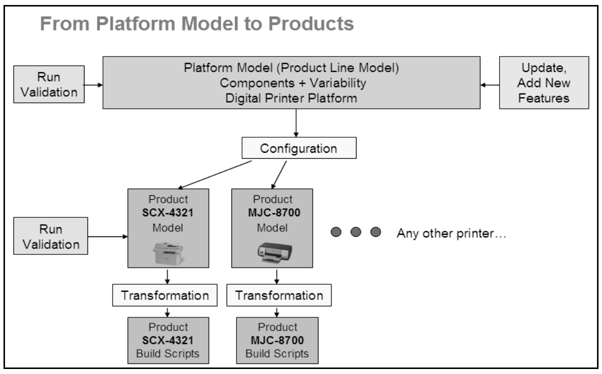

From Platform Model to Products
Run Validation
Platform Model (Product Line Model) Components + Variability Digital Printer Platform
Update, Add New Features
Configuration
Product SCX-4321 Model
Product MJC-8700 Model
Run Validation
Any other printer…
Transformation
Transformation
Product SCX-4321 Build Scripts
Product MJC-8700 Build Scripts

Fig. 5. Product definition process

The process of using our tool is depicted in Fig. 5. The domain engineer models the product line platform which contains hierarchic structure of components and their connections together with variation points. Fig. 6 shows an example of the internal structure of a component with variation points of type *optional*. To continue the process, validation checks can be activated on the domain model. The next step is the generation of a product model by configuring the product line model. This can be done in several steps using stage configuration. The last step is the generation of product artefacts for the product model. Whenever there is an update in the product line definition, the domain engineer can go back to the product line model, update it and repeat the validation, configuration and build script generation steps.

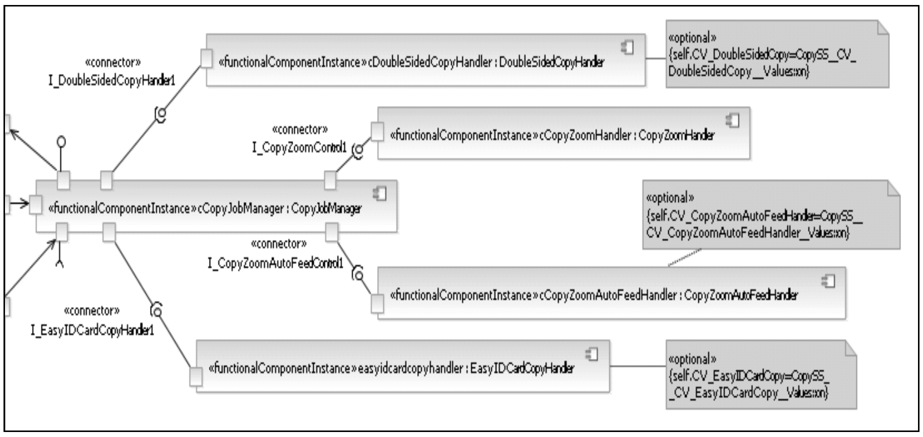

«connector»
I_DoubleSidedCopyHandler1
«functionalComponentInstance» cDoubleSidedCopyHandler : DoubleSidedCopyHandler
«optional»
{self.CV_DoubleSidedCopy=CopySS_CV_DoubleSidedCopy__Values::on}
«connector»
I_CopyZoomControl1
«functionalComponentInstance» cCopyZoomHandler : CopyZoomHandler
«functionalComponentInstance» cCopyJobManager : CopyJobManager
«optional»
{self.CV_CopyZoomAutoFeedHandler=CopySS_CV_CopyZoomAutoFeedHandler_Values::on}
«connector»
I_CopyZoomAutoFeedControl1
«functionalComponentInstance» cCopyZoomAutoFeedHandler : CopyZoomAutoFeedHandler
«connector»
I_EasyIDCardCopyHandler1
«functionalComponentInstance» easyidcardcopyhandler : EasyIDCardCopyHandler
«optional»
{self.CV_EasyIDCardCopy=CopySS_CV_EasyIDCardCopy__Values::on}

Fig. 6. A component with variability

Our model-driven development environment for software product lines extends the IBM Rational toolset, which is built using the Eclipse framework. IBM Rational tools contain an out-of-the-box environment for model driven development in UML, including editors, palettes, import, export, code generation.
The issues addressed by this work can be applied to various industries such as electronics, automotive, aerospace and defense, and more.

4 Model-based Development of Telecom Services

Modeling can be an effective way to manage the complexity of service-oriented software development. Although it is widely used, its adoption in the Telecommunications domain is not yet widespread.

Telecom service development requires deep low-level knowledge of numerous protocols (e.g. SIP, SDP, Diameter, HTTP, SOAP) and on how to synchronize between them. In IP telephony, several platforms and tools for service development are available which simplify the service development process to some extent, but still they don't hide from their users all protocol details.

Our Telecom Service Creation Environment (TSCE) enables designers to create models of such services without requiring any knowledge of the underlying protocols and to generate runnable services from them without additional code being written. It allows designers to design services using the Telecom Service Domain Specific Language (TS-DSL) and provides a level of automation for implementing the service design methodology we adopted. Part of the TS-DSL meta-model can be seen in Figure 1.

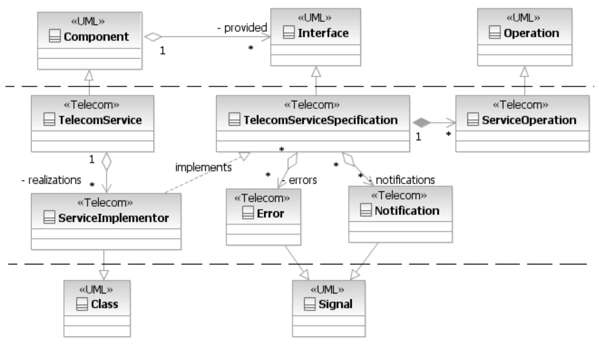


Fig. 7. Telecom Service Meta-model fragment

TS-DSL hides the internals of the protocols focusing on their provided functionality and provides high level building blocks for the design of services. TS-DSL also support invocations of external services of diverse types (e.g. SIP Services, Web Services) through different protocols or mechanism (e.g. SIP, HTTP, SOAP) - hiding the underline invocation and synchronization details from the developers. TS-DSL also leverages the Shared Information/Data models introduced by Telecommunication Management Forum.

TSCE is implemented on top of Rational tools (RSA/RSM), which facilitate the reuse of existing services, use of templates, behavioral patterns, and pre-defined service elements to speed up the design and implementation process. TS-DSL is implemented in TSCE using a UML profile and a telecom model library.

The model library introduces a set of predefined entities that encapsulate data and functionality widely-used in Telecom service models. For example: ***Party, Call, Instance Message***, ***and Biller.*** Developers can use these entities and invoke their operations in their model.

TSCE introduces a set of views, actions, service creation templates, and TSCE-related palettes to simplify domain-specific language element creation and manipulation (see Figure 2).

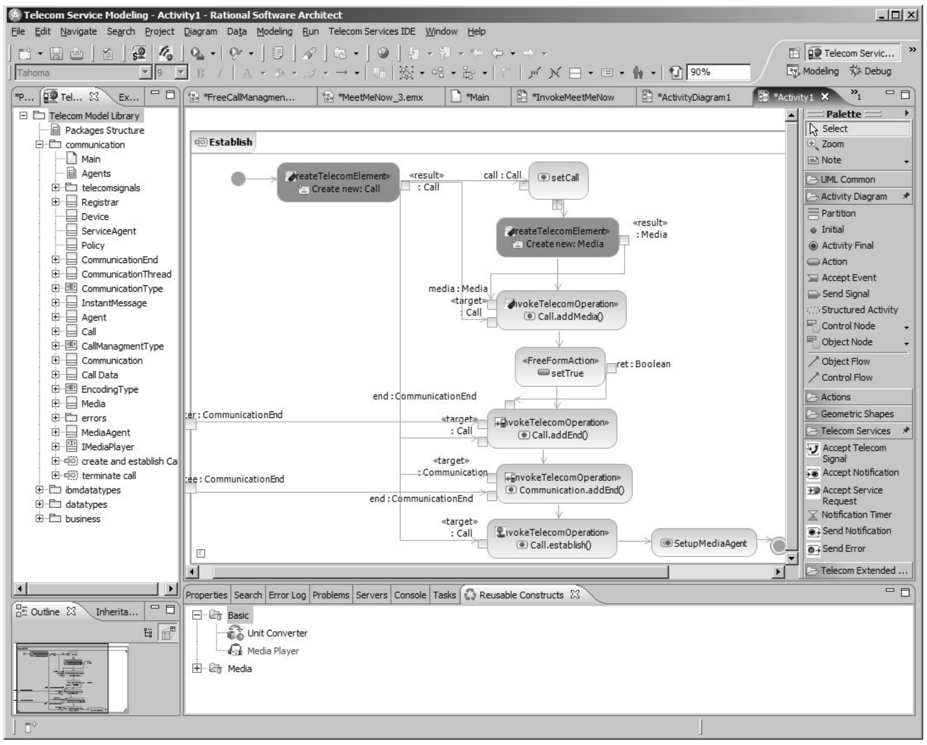

Telecom Service Modeling - Activity1 - Rational Software Architect
Telecom Model Library
Establish
«createTelecomElement» Create new: Call
«result» : Call
call : Call
setCall
«createTelecomElement» Create new: Media
«result» : Media
media : Media
«target» : Call
«invokeTelecomOperation» Call.addMedia()
«FreeFormAction» setTrue
ret : Boolean
end : CommunicationEnd
«invokeTelecomOperation» Call.addEnd()
«invokeTelecomOperation» Communication.addEnd()
«invokeTelecomOperation» Call.establish()
SetupMediaAgent
Palette
Reusable Constructs
Unit Converter
Media Player

Fig. 8. The Telecom Service Creation Environment

The behavior of a telecom service is specified by state machines and activity diagrams. Each telecom service is created with a main state machine which specifies the service interaction with a client. The designer can add “entry”, “do”, and/or “exit” activities for each state to define the logic to perform on state initialization, processing, and just before exiting it. Figure 3 contains the main state machine of the “Free Call with Ads” service we developed using TSCE.

State activities can be seen as entry points to the service activity flow. The flow can invoke a variety of UML and TS-DSL actions and control nodes connected via control flow links. Data flow is defined via data links between the action’s pins. Actions can invoke instance creations, operation calls, external service invocations, other activities, etc. For example, Figure 2 contains an activity diagram that creates a Call and establishes it between an invitee and inviter with no protocol dependencies.

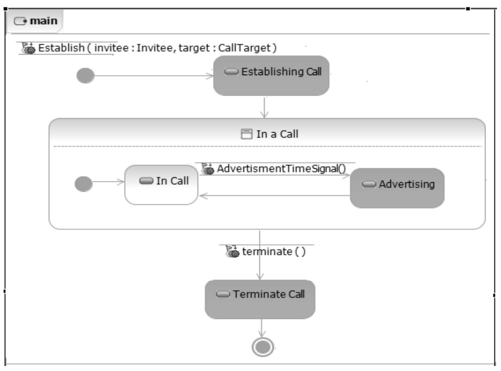
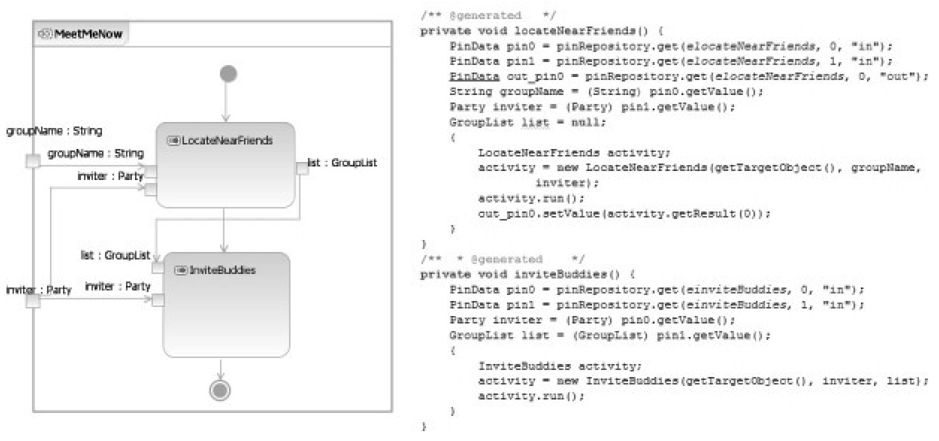


Fig. 9. Free Call with Ads example: Main State Machine chart

One of the main challenge of this work was in defining the right level of abstraction that is both powerful enough to describe all the required functionality of a service and yet enables generation of runnable service code, i.e. defining mapping rules and infrastructure that close the abstraction gap and transform the designed services into running code.

TSCE transformation extends Rational Software Architect's built in transformation from UML to Java and adds our service specific rules to perform the task of generating both the static (e.g. SIP Servlet structure, sip.xml) and behavioral parts of the service. Figure 4 shows a fragment of the generated code for "Meet Me Now" activity.



```
/** @generated   */
private void locateNearFriends() {
    PinData pin0 = pinRepository.get(elocateNearFriends, 0, "in");
    PinData pin1 = pinRepository.get(elocateNearFriends, 1, "in");
    PinData out_pin0 = pinRepository.get(elocateNearFriends, 0, "out");
    String groupName = (String) pin0.getValue();
    Party inviter = (Party) pin1.getValue();
    GroupList list = null;
    {
        LocateNearFriends activity;
        activity = new LocateNearFriends(getTargetObject(), groupName,
                inviter);
        activity.run();
        out_pin0.setValue(activity.getResult(0));
    }
}
/**  * @generated    */
private void inviteBuddies() {
    PinData pin0 = pinRepository.get(einviteBuddies, 0, "in");
    PinData pin1 = pinRepository.get(einviteBuddies, 1, "in");
    Party inviter = (Party) pin0.getValue();
    GroupList list = (GroupList) pin1.getValue();
    {
        InviteBuddies activity;
        activity = new InviteBuddies(getTargetObject(), inviter, list);
        activity.run();
    }
}
```

Fig. 10. Generated Code Sample

Our approach radically simplifies service design, cuts down service time-to-market, makes the service design process accessible to designers not familiar with telephony protocol details, and simplifies maintenance.

References

1. IBM Haifa Research Lab: http://www.haifa.ibm.com/
2. IBM Haifa Research Lab – Model Driven Engineering Technologies group: http://www.haifa.ibm.com/dept/services/mdet.html
3. IBM Tokyo Research Lab: http://www.trl.ibm.com/extfnt_e.htm
4. Dotan D., Kirshin A.: Debugging and Testing Behavioral UML Models. OOPSLA Companion 2007, pp. 838–839. ACM Press (2007)
5. http://www.eclipse.org/tptp
6. Kawahara R., Nakamura H., Dotan D., Kirshin A., Sakairi T., Hirose S, Ono K., Ishikawa H.: Verification of embedded system's specification using collaborative simulation of SysML and Simulink models, MBSE 2009

Applying Model Driven Approach to parallel platform architecture

Irv Badr, IBM Rational, USA

Abstract. Flexibility, scalability, power consumption, and in the case of consumer devices, an optimized user-experience are main requirements posed by tomorrow's real-time and embedded systems. System scalability and flexibility are key factors in fast-time-to-market and allow manufacturers and service providers to be competitive. Executable models and virtualization techniques become a vital part of the workflow used in software development for scalable parallel hardware architectures

Model-driven reverse engineering of COBOL-based applications

Franck Barbier*†, Sylvain Eveillard*, Kamal Youbi* and Eric Cariou†

*Netfective Technology, OMG member, SOA consortium member

†University of Pau

1. Introduction

Model-Driven Development (MDD) [1] is nowadays a proven paradigm for constructing cost-effective applications. Their design cycle satisfies time-to-market constraints. Inspired by object-orientation (OO programming languages, OO databases, OO analysis and design methods…), MDD favors software quality in general, *i.e.*, reusability and maintainability. Models[1], which are first-class components in development processes, also greatly favor portability. More precisely, *Platform-Independent Models* (PIMs) are free from technical concerns, while *Platform-Specific Models* (PSMs), derived from PIMs by means of transformations, are annotated with platform-oriented configuration information to generate end-user applications.

To be able to cope with large-scale MDD applications, the demand for appropriate MDD tools has increased. These tools support the management of sizeable modular models through repositories. They also offer conformance with and adaptation to standards like the Eclipse Modeling Framework (EMF). They can generate code, transform source models into target models and are able to define new metamodels to build Domain-Specific Modeling Languages (DSMLs).

For a long time, MDD tools have supported reverse engineering functionalities. These features are often limited to code parsing. The results are cartographic views of this code. This can even be useful for code that does not come from model transformation. For obvious reasons, this code is usually OO-based (e.g., reversing Java code), because only with great difficulties can non OO code be reformatted into the form of models. For code generated from models, tools have difficulties guaranteeing a kind of synchronization between models and code. This is especially true when programmers intend to make changes directly in the source code2. Moreover, MDD has the reputation of being appropriate for designing applications from scratch. Nowadays, applications built from models exist, but there is often a divergence between model states and code shape. This results from manual alterations.

More generally, the great majority of applications are not derived from models. The spreading and success of models' technology thus strongly rely on the ability of this

[1] We consider UML (*Unified Modeling Language*) models.

[2] Performance tuning has often been a primary reason to intervene directly on code.

technology to integrate existing information systems in a seamless way. Model-driven reverse engineering may opportunely play this role.

To that extent, his paper discusses a method and a tool that provide an integrated reverse engineering framework, which is plugged into Eclipse. The tool named BLU AGE®3 has the ability to generate applications from models without writing a single line of code. In BLU AGE®, PIMs are constructed by software analysts while technical cartridges (PDMs standing for Platform Description Models) are designed by software architects. BLU AGE® is able to transform PIMs into PSMs and to link these PSMs to PDMs. This paper explores how PIMs can be, as much as possible, automatically generated from legacy systems. We focus on COBOL-based applications.

2. Goals

The key goals of this paper are to address:

- Non OO code issues. We discuss the reverse engineering of common[4] COBOL in a specific context. Precisely, this code is written in COBOL, but conforms to the IBM VisualAge PACBASE (VAP) tool. The code structuring thus respects particular rules and formats. Concretely, VAP programs are written by means of a superset (a dialect) of the COBOL language itself. Programs are also equipped with helpful VAP-based configuration data linked to legacy technology (*e.g.*, character screens). However, one must remember that this tool comes from programs created in the eighties, so code complies to the standards and the structured programming spirit of that time. That is why there is high redundancy; This is due to the ignorance of the OO approach at that time. So, the difficulty in converting such legacy information systems into models remains high, especially OO models.
- Semantic issues. Simply having a cartographic view of old COBOL is not enough. So models are not only partial or complete viewpoints of legacy systems, but also include a re-formalization of the business rules engraved in COBOL programs. As a result, models obtained from reverse engineering are intended to become the inputs of the generation processes, which are used to rebuild applications based on today's platforms. This is mainly Java EE and .NET, but also includes their associated components: Struts, JavaServer Faces (JSF), Hibernate, Java Persistence API (JPA), Windows Presentation Foundation (WCF)… About semantic issues, this paper also sketches the alignment of the proposed reverse engineering process with KDM (Knowledge Discovery Meta-Model), an OMG standard [2].
- Systematic and automated reverse engineering issues. As far as possible, reverse engineering should never be done manually. Although human intervention is inevitable, a tool is required to support most of the tedious tasks caused by sizeable volumes. To that extent, we present the REVERSE component of BLU AGE®, an Eclipse/EMF-based CASE (*Computer-Aided Software Engineering*)

[3] www.bluage.com
[4] The proposed approach is not dedicated to object-oriented COBOL but ordinary COBOL.

tool that enables the elaboration of PIMs that are endowed with context-specific annotations (a.k.a. stereotypes in UML). These PIMs result from the extraction of legacy code. The case study described in this paper is the first COBOL benchmark of the REVERSE module of BLU AGE®. Until now, the perfecting of this module has occurred by means of Java only. The results exposed in this paper show the need for a more open and sophisticated reverse engineering process and tool due to, essentially, the non OO nature of COBOL and VAP. Lessons learned also comment on the large-scale factors of this experiment and its economical facets.

3. Modernization context

VisualAge PACBASE is an application generator. Billions of lines of COBOL exist all over the world, which have been produced with this environment. For historical reasons, such applications require specific execution contexts, namely old terminals (non graphical window-based screens), mainframes and CICS (*Customer Information Control System*)… Regarding CICS, VAP legacy systems mix business logic and CICS commands. So, an important challenge is to first separate both aspects to converge with the "PIMs versus PSMs" philosophy. The next challenge is to substitute CICS for another execution platform, like a Java EE server for example.

Despite its recognized qualities, the VAP integrated development environment raises critical problems:

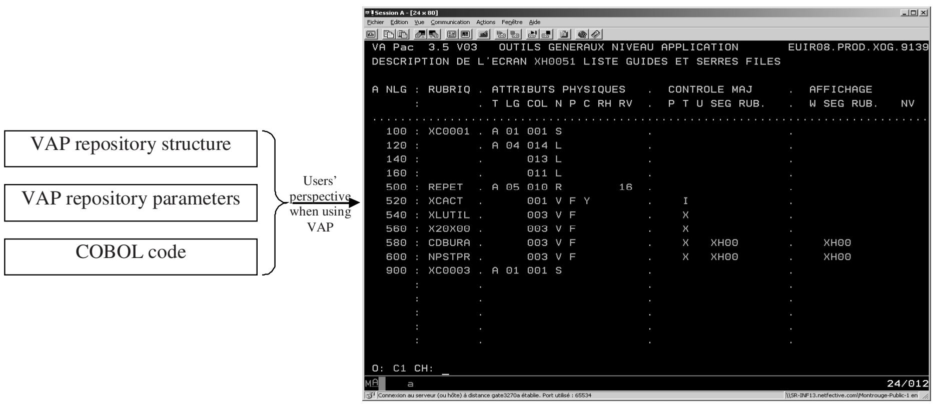
- The environment, and thus its resulting applications, will be maintained by IBM up to 2015;
- It does not support any kind of interoperability with modern non proprietary platforms;
- Expert users exist[5], but advances in computer and software technologies create difficulties and even risks, when relying on non durable adequate competencies;
- Applications based on this environment contain an important amount of business knowledge. The need for extracting this knowledge in a readable and usable way is a crucial issue in investment perenniality;
- Maintenance costs cannot be cut down and are high. This is due to the significant dependency upon external assistance and support.

4. Modernization approach

Given that an application is based on various old software technologies, the overall goal of modernization is *a priori* to replace this application by another equivalent one (in terms of functions offered to users). One of the benefits of having a renewed application, based on Java EE for instance, is the gain in technical agility provided by Java EE (Web integration abilities, load balancing, ease and transparency of service access and distribution, compliance with standards, Java EE product (server)

[5] www.napug.com

independence…). Of course, using MDD for reverse engineering provides the opportunity to change and/or to extend the set of functions corresponding to enhanced user requirements. Models (PIMs) formally express these requirements. PSMs linked to Java EE for example, may thus be derived from these PIMs, which are outputs of the reverse engineering process themselves.


VAP repository structure
VAP repository parameters
COBOL code
Users' perspective when using VAP
Session A - [24 x 80]
Fichier Edition Vue Communication Actions Fenêtre Aide

```
VA Pac  3.5 V03   OUTILS GENERAUX NIVEAU APPLICATION        EUIR08.PROD.XOG.9139
DESCRIPTION DE L'ECRAN XH0051 LISTE GUIDES ET SERRES FILES

A NLG : RUBRIQ . ATTRIBUTS PHYSIQUES   .  CONTROLE MAJ     .  AFFICHAGE
      :        . T LG COL N P C RH RV  .  P T U SEG RUB.   .  W SEG RUB.   NV
................................................................................
  100 : XC0001 . A 01 001 S            .                   .
  120 :        . A 04 014 L            .                   .
  140 :        .      013 L            .                   .
  160 :        .      011 L            .                   .
  500 : REPET  . A 05 010 R        16  .                   .
  520 : XCACT  .      001 V F Y        .    I              .
  540 : XLUTIL .      003 V F          .    X              .
  560 : X20X00 .      003 V F          .    X              .
  580 : CDBURA .      003 V F          .    X   XH00       .    XH00
  600 : NPSTPR .      003 V F          .    X   XH00       .    XH00
  900 : XC0003 . A 01 001 S            .                   .
      :        .                       .                   .
      :        .                       .                   .
      :        .                       .                   .
      :        .                       .                   .
      :        .                       .                   .

O: C1 CH: _
MA     a                                                                  24/012
Connexion au serveur (ou hôte) à distance gate3270a établie. Port utilisé : 65534    \\SR-INF13.netfective.com\Montrouge-Public-1 en
```

Figure 1. VAP organization

So, three phases are distinguished:

1. The reverse engineering process itself. We describe in this paper some implementation principles and details of the REVERSE component of BLU AGE®, especially the generation of text based on a semi-natural language. This language shares a common metamodel (*i.e.*, "it is an instance of") with a prefabricated metamodel, which is dedicated to the way people may use VisualAge PACBASE (Figure 1). In fact, programs are described in a COBOL-like language that is associated with screen layout/control information, and so on. All of this material can be recorded as an instance of a metamodel. In this respect, reverse operations intensively use model transformation technologies. More precisely, the *Atlas Transformation Language* (ATL)[6] instruments these reverse actions.
2. The validation phase. Models can be used to generate the new application. This means that all functions and business rules in legacy code must be properly captured. This phase is called validation. Once the PIMs are viewable as outputs of the reverse process, one may wonder if they are the trustworthy representation of the existing system. Test data and scenarios established from the existing system can be replayed for the renewed application. By means of another component (named BUILD), the testing of models is carried out within Eclipse. This is done with the help of the new generated code which is synchronized to models, even graphically. In this respect, Figure 2 shows a UML model that represents screen linking and data processing. On the right hand side, there is a UML activity, which describes a given screen. Several scenarios exist to move from this screen to others (on the left hand side) through an intermediate column (called "swimlane" in UML jargon). In this column, activities represent data processing in servers (queries for

[6] www.eclipse.org/m2m/atl/

example). Hence, testing facilities allow one to assign breakpoints to activities. One can carry out debugging step by step. This is how validation may succeed provided that test data is rich and significant enough.

3. The measurement activity. The reverse modeling process has to be proven economically relevant. However, demonstrating that PIMs are trustworthy and complete pictures of the current functions is time-consuming. While elementary operations are treated automatically, complex operations are progressively controlled and scored[7]. By hand first, these operations are annotated with technical and/or functional markers. The reverse process captures the available knowledge to avoid as much as possible to request further work from reverse engineers. In this scope, the pilot project (see next section) aims at anticipating the scalability of the proposed modernization process; if it can be generalized especially. In this paper, we do not focus on the re-generation of the existing application to target modern platforms. If the targeted application's PIMs are consistent and complete, rebuilding a new system with BLU AGE® is quite simple compared to computing these PIMs by means of reverse engineering (phases 1 and 2). Moreover, it can be done in a cost-effective way (phase 3). That is why projected man-month charges have been established for the pilot project. So, effective man-month consumption should be compared to these forward-looking costs. This paper describes this experiment which has just finished. At this time, the feasibility and the applicability of this reverse engineering process have been demonstrated on a medium scale and in a cost-effective way.

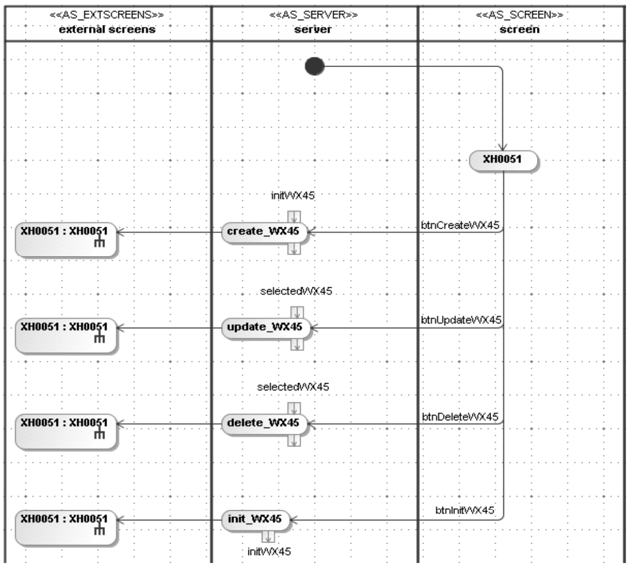


Figure 2. Example of a UML dynamic model coming from the reverse engineering process

[7] The number of elementary statements used like GOTO, COBOL module performed (PERFORM command) subprograms called… are established.

5. Case study description

Intermarché is one of the main European companies in the retail sector. STIME is its information system division. The SCAFRUITS application is concerned with the provisioning of fruits and vegetables from referenced suppliers, as well as the sale and distribution of these items to franchised outlets all over Europe. Shops interact by means of the SCAFRUITS application to have the best products at the best prices under the best conditions (short time delivery, sanitary quality assurance, synchronization with volatile demand…).

The business scope of SCAFRUITS is broad: order management, shipping, supplier and product qualification and referencing, timely price management, product activation/inhibition… For obvious common reasons, the application is also composed of many facility programs and sub-programs, which play the role of transversal supports for the business-oriented primary use cases.

The application's design and initial utilization began in 1994. It has continuously evolved since that time. Today, from a business viewpoint, the application is composed of 85 Transactional Processing (TP) and 23 batch processes. A TP is a typical interactive use case, *e.g.*, a user carries out the referencing of a fruit supplier through an old-fashion screen to finally be notified about the success (or possible failure) of its operation. Additionally, transversal facility subprograms are used for checking the conformity of products to EU sanitary standards.

Concerning its technical facets, the size of the application is estimated to be equal to 3M of LoC, 600 programs, 400 screens, 200 batch programs, 300 potential users, 48,000 product references with only 2,000 active references at a time. There are 350,000 transactions per day and 100,000 created order lines per day. The savings, if the full application can be moved from VAP to a modern platform like Java EE, are estimated at around € 1.5M per year.

The pilot project, called VAP2BA (VisualAge PACBASE to BLU AGE®), has a scope-limited functional perimeter. This consists of product referencing and ordering, including CRUD (*Create, Read, Update, Delete*) operations. The pilot project is concerned with 15 TP and 15 batch (business) processes. Here are some of the use cases, which are part of the pilot project:

- supplier order creation;
- supplier order reception;
- Etc.

Test scenarios and data rely on these use cases. In fact, it amounts to collecting data. This data aims at being restructured into objects, in the OO sense. More precisely, these objects and their links are instances of the classes and associations (see Figure 10) found in the PIMs, which are outputs of the reverse engineering process. Once established, testing occurs through scenarios, *e.g.*, a screen linking is required for playing the "supplier order reception" use case.

This global activity is critical to demonstrate that the PIMs are "correct" (*i.e.*, "The validation phase" above).

6. Method [3]

The technical approach is based on the *PACBASE Access Facility* (PAF) component of VAP. VAP developers use the VAP user interface (Figure 1) to build their applications by means of a COBOL-like language, while adhering to VAP design guidelines. This amounts to taking into account numerous flat files with a complex specific organization (redundancy, implicit dependency among files based on implicit string matching, etc.). Practically speaking, PAF offers a SQL-like language that can access data in the form of VAP repository entities.

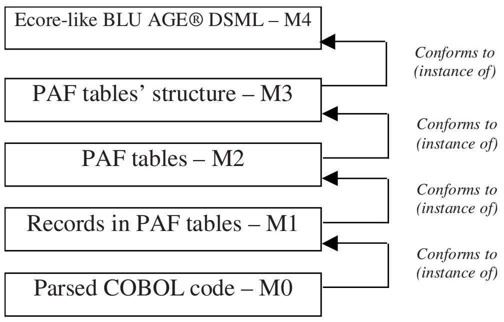

Ecore-like BLU AGE® DSML – M4
PAF tables' structure – M3
PAF tables – M2
Records in PAF tables – M1
Parsed COBOL code – M0
Conforms to (instance of)
Conforms to (instance of)
Conforms to (instance of)
Conforms to (instance of)

Figure 3. Metamodel layering

The proposed method intensively relies on predefined metamodels (Figure 3). It starts from an Ecore-like DSML[8]. Ecore (Figure 4) is the Eclipse metalanguage for coping with any kind of model. The structure of the PAF tables is thus represented by means of the Ecore-like metalanguage. The result is that the complex organization of VAP is captured once and for all[9]. The logic of VAF relies on metatypes, which are members of this metamodel (M3 level): *Role*, *Table*[10], *Column*, *Library* (of code), *Session*, *User…* It also relies on associations between these metatypes. For example, a role is composed of zero or *n* tables; a table is composed of at least one and at most *n* columns. For example, the *Table* metatype conforms by definition to a BLU AGE® metatype specifically designed for reverse engineering.

[8] Domain-Specific Modeling Language.

[9] Section *Lessons learned from the case study* discusses why this solution is not satisfactory in forthcoming developments of BLU AGE®.

[10] In the VAP logic, a table is a unit of organization. For example, a table may describe a batch process.

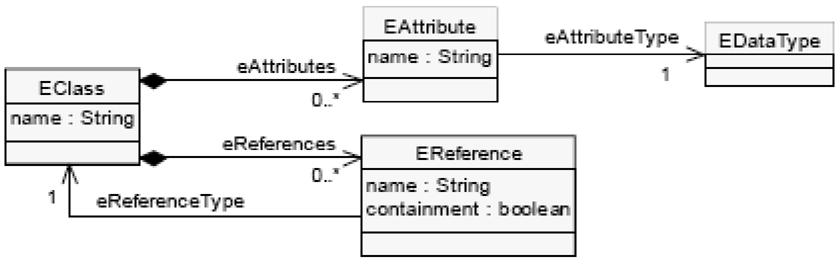


Figure 4. Ecore core metatypes in Eclipse

Another metamodel (M2 level in Figure 3) is application-dependent. A SCAFRUITS table **structure** is an instance of the *Table* metatype at the M3 level. For instance, the *EC02* table structure in SCAFRUITS has five columns, each having a given size, format and label… Copies of the *EC02* table itself are thus located at the M1 level as instances of the *EC02* table structure.

So, PAF is used to populate records at the M1 level. Data samples extracted by means of PAF are described in Figure 5.

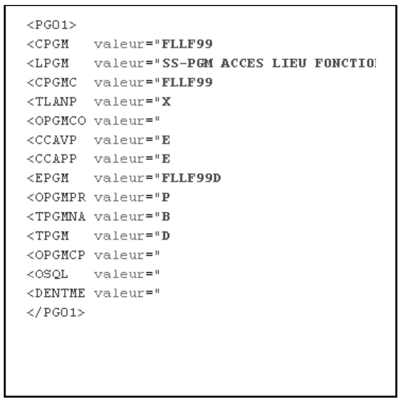

```
<PG01>
<CPGM   valeur="FLLF99
<LPGM   valeur="SS-PGM ACCES LIEU FONCTIO
<CPGMC  valeur="FLLF99
<TLANP  valeur="X
<OPGMCO valeur="
<CCAVP  valeur="E
<CCAPP  valeur="E
<EPGM   valeur="FLLF99D
<OPGMPR valeur="P
<TPGMNA valeur="B
<TPGM   valeur="D
<OPGMCP valeur="
<OSQL   valeur="
<DENTME valeur="
</PG01>
```

Figure 5. Rough XML data resulting from PAF queries

Since extractions are composed of several XML files, removing useless (parasite) data and redundancies is a primary issue. For example, a TP may use the *SE01* segment, while a batch process uses the same segment. This segment must appear once and for all in an appropriate model.

The PAF extraction model is intentionally used to first eliminate redundancy. Next, several ATL transformations are run in sequence to explicitly re-create the dependencies between VAP entities (Figure 6).

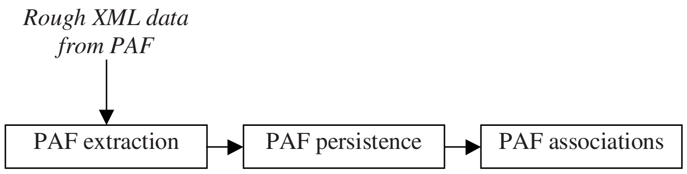
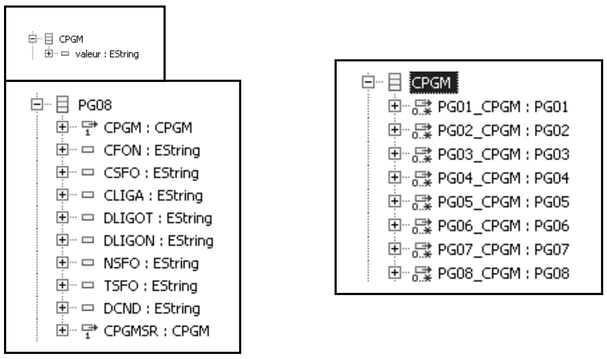


Figure 6. VAP information processing

In Figure 6, the PAF persistence model is computed by means of an ATL transformation from the PAF extraction model. The same applies to the PAF associations, which adds meaningful inverse references to the VAP entities that come from the PAF persistence model. The PAF persistence model re-creates references (instances of *EReference* in Figure 4) from flat files. The PAF associations is composed of inferred inverse references from the the PAF persistence model.

As an illustration, each VAP entity (a batch process for example) has a main screen (*PG01* for example) and some possible screens. Each COBOL statement is registered in a VAP entity, say *PG08*. One must then recompose the link between *PG01* and *PG08* in the models. In VAP however, string matching and cross-references between files are the only way to determine dependencies.



```
CPGM
  valeur : EString

PG08
  CPGM : CPGM
  CFON : EString
  CSFO : EString
  CLIGA : EString
  DLIGOT : EString
  DLIGON : EString
  NSFO : EString
  TSFO : EString
  DCND : EString
  CPGMSR : CPGM

CPGM
  PG01_CPGM : PG01
  PG02_CPGM : PG02
  PG03_CPGM : PG03
  PG04_CPGM : PG04
  PG05_CPGM : PG05
  PG06_CPGM : PG06
  PG07_CPGM : PG07
  PG08_CPGM : PG08
```

Figure 7. PAF persistence and PAF associations samples

Following this logic, an instance of the PAF associations model is shown on the right hand side of Figure 7. The *CGPM* batch process (a VAP entity) supports navigation to all of its linked *PG01* objects and to all of its *PG08* objects, etc. In short, the PAF associations model complements the PAF persistence model by supplying reverse navigability.

The reverse engineering method is in fact divided into three phases:

- The extraction phase is fully generic (*i.e.*, independent of the SCAFRUITS application itself) and fully automated. This phase is built onto the metamodels in Figure 3. It contributes to having a nice and rich cartographic view of a VAP application within Eclipse. This phase is based on the above described transformation processes;
- The interpretation phase is semi-automated in the sense that it depends upon the deep natures of the legacy technology and business application. The COBOL-like code (Figure 8) is parsed and transformed into a semi-natural

language[11] (Figure 9). This code conforms to a predefined metamodel. One key advantage of the proposed method is that the semi-natural language in Figure 9 conforms to a meta-model, which is an ATL transformation of the former. To more or less automatically generate business rules in a neutral formalism, reverse engineers provide on-the-fly information on (1) what are the means, constraints and rules of the legacy technology to manage business rules and, (2) what are and where are the true business rules of the reversed application itself. As a result, the semi-natural language sentences are a great help for readability and comprehensibility for both the application and its support technology. Outputs allow the direct generation of BLU AGE® stuff. This holds true for even OCL (*Object Constraint Language*) constraints, which are in charge of representing business rules in BLU AGE®.

•

```
SousFonction 'FERMETURE DU FICHIER RFTZ93WL' F2OSFPA {
    IF(Z499-XCF06A = SPACES AN Z499-XCF07A = SPACES) {
        MOVE  5-PA00-CPTENR    Z499-X09929
        MOVE  'LUS'    Z499-X07X00
        MOVE  'RFTZ93WL'    Z499-X08X18
        MOVE  'FI'    Z499-XCF02A
        PERFORM F8A
        MOVE  SPACES    Z499-XCF02A
        CLOSE PA-FICHIER
    }
}
```

Figure 8. VAP-oriented COBOL

```
Code de la sous-fonction "FERMETURE DU FICHIER RFTZ93WL" , nom technique :  F2OSFPA {
  Si la Rubrique XCF06A du Segment Z499 vaut "" et la Rubrique XCF07A du Segment Z499 vaut ""
  faire
    Déplacer le compteur de la Rubrique CPTENR du Segment PA00 dans la Rubrique X09929 du Segment Z499
    Déplacer "LUS" dans la Rubrique X07X00 du Segment Z499
    Déplacer "RFTZ93WL" dans la Rubrique X08X18 du Segment Z499
    Déplacer "FI" dans la Rubrique XCF02A du Segment Z499
    Exécuter le code de la fonction ou de la sous-fonction F8A
    Déplacer "" dans la Rubrique XCF02A du Segment Z499
    CLOSE la Rubrique FICHIER du Segment PA
  finfaire
}
```

Figure 9. Semi-natural language (in French) generated from VAP-oriented COBOL

- The publication phase leads to view, transform, but mostly, refactor, the material coming from the interpretation into BLU AGE® metatypes (Figure 10). However, all of the COBOL can be easily and straightforwardly managed within Eclipse (Figure 11, right hand side). At this time, there is a Java-like language, which is offered to users to translate COBOL-oriented models into this language (*i.e.*, on the right hand side of Figure 11). At this time, the mapping with entities, business objects, controllers (in BLU AGE® jargon) is, for efficiency reasons, carried out with a certain strategy.

[11] For example, the French word "Déplacer" means "to move" in English. Generally speaking, people are more confident when reading text close to their native language.

As an ongoing implementation, the inclusion of a representation phase in the reverse engineering process is described in the next section.

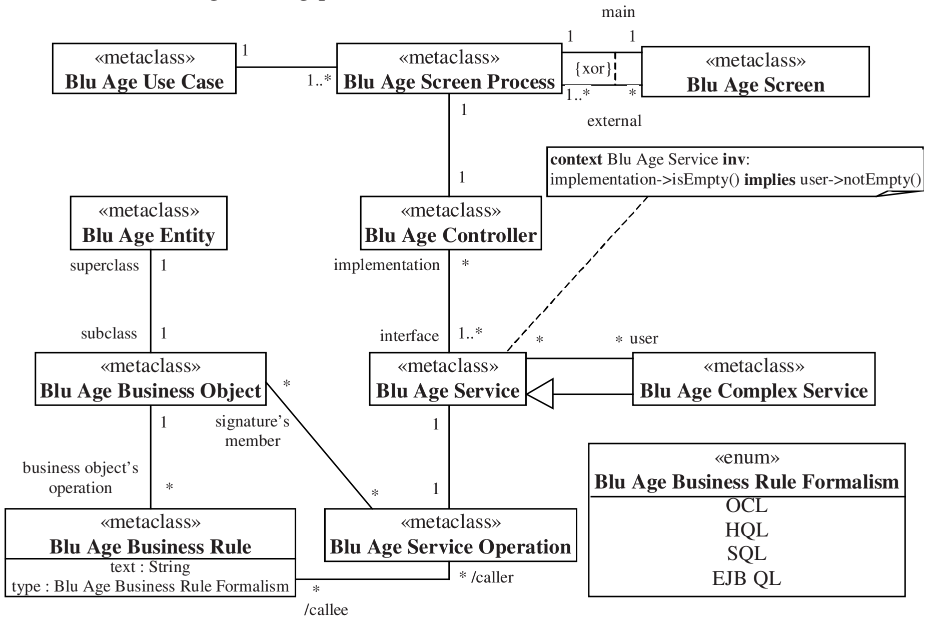
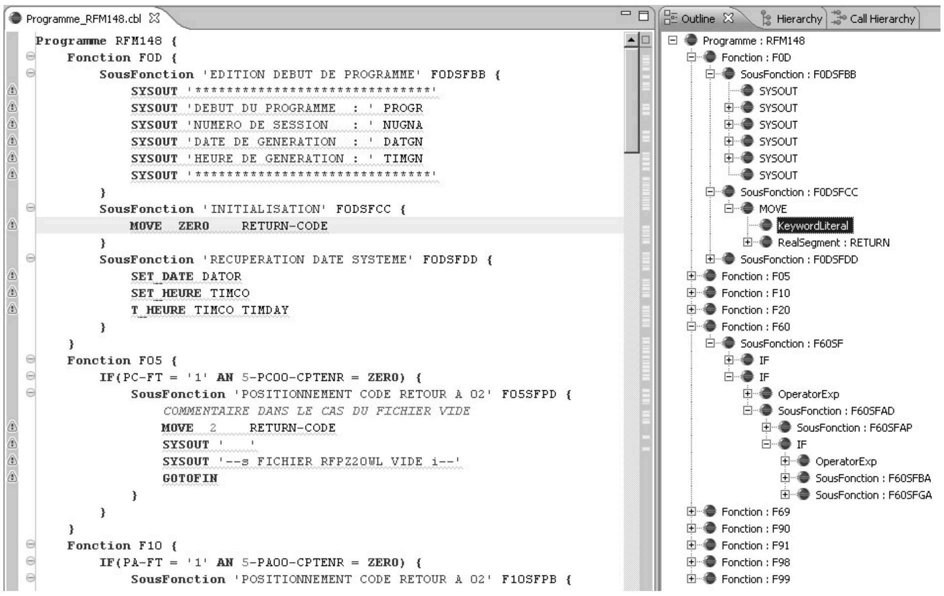

«metaclass» Blu Age Use Case
«metaclass» Blu Age Screen Process
«metaclass» Blu Age Screen
main
external
{xor}
1
1..*
«metaclass» Blu Age Controller
context Blu Age Service inv: implementation->isEmpty() implies user->notEmpty()
«metaclass» Blu Age Entity
superclass
subclass
implementation
interface
user
«metaclass» Blu Age Business Object
«metaclass» Blu Age Service
«metaclass» Blu Age Complex Service
signature's member
business object's operation
«enum» Blu Age Business Rule Formalism
OCL
HQL
SQL
EJB QL
«metaclass» Blu Age Business Rule
text : String
type : Blu Age Business Rule Formalism
«metaclass» Blu Age Service Operation
* /caller
* /callee

Figure 10. BLU AGE® metatypes



```
Programme RFM148 {
    Fonction FOD {
        SousFonction 'EDITION DEBUT DE PROGRAMME' FODSFBB {
            SYSOUT '******************************'
            SYSOUT 'DEBUT DU PROGRAMME  : ' PROGR
            SYSOUT 'NUMERO DE SESSION   : ' NUGNA
            SYSOUT 'DATE DE GENERATION  : ' DATGN
            SYSOUT 'HEURE DE GENERATION : ' TIMGN
            SYSOUT '******************************'
        }
        SousFonction 'INITIALISATION' FODSFCC {
            MOVE  ZERO    RETURN-CODE
        }
        SousFonction 'RECUPERATION DATE SYSTEME' FODSFDD {
            SET_DATE DATOR
            SET_HEURE TIMCO
            T_HEURE TIMCO TIMDAY
        }
    }
    Fonction F05 {
        IF(PC-FT = '1' AN 5-PC00-CPTENR = ZERO) {
            SousFonction 'POSITIONNEMENT CODE RETOUR A 02' F05SFPD {
                COMMENTAIRE DANS LE CAS DU FICHIER VIDE
                MOVE  2    RETURN-CODE
                SYSOUT '    '
                SYSOUT '--s FICHIER RFPZ20WL VIDE i--'
                GOTOFIN
            }
        }
    }
    Fonction F10 {
        IF(PA-FT = '1' AN 5-PA00-CPTENR = ZERO) {
            SousFonction 'POSITIONNEMENT CODE RETOUR A 02' F10SFPB {
```

```
Programme : RFM148
    Fonction : F0D
        SousFonction : F0DSFBB
            SYSOUT
            SYSOUT
            SYSOUT
            SYSOUT
            SYSOUT
            SYSOUT
        SousFonction : F0DSFCC
            MOVE
                KeywordLiteral
                RealSegment : RETURN
        SousFonction : F0DSFDD
    Fonction : F05
    Fonction : F10
    Fonction : F20
    Fonction : F60
        SousFonction : F60SF
            IF
            IF
                OperatorExp
                SousFonction : F60SFAD
                    SousFonction : F60SFAP
                    IF
                        OperatorExp
                        SousFonction : F60SFBA
                        SousFonction : F60SFGA
    Fonction : F69
    Fonction : F90
    Fonction : F91
    Fonction : F98
    Fonction : F99
```

Figure 11. COBOL code management within Eclipse (right hand side)

7. Lessons learned from the case study

The first phase (the extraction phase) required 4 man-months (2 persons). This was to design and perfect the reverse engineering method itself; while making slight implementation amendments in the CASE tool possible. In order to obtain appropriate knowledge, support and any kind of access to the SCAFRUITS application, a ½ man-month support was provided by STIME. Minor modifications of the REVERSE component of BLU AGE® occurred during this first stage. The unique result of this phase was a beta "industrial" version of this component. The application-independent nature of the process leads to success, even though a functional subset of SCAFRUITS was used for the experimentation.

The second phase is an ongoing phase (3 months). This phase covers the interpretation and publication phases. This last phase includes BLU AGE® training: the construction within Eclipse of BLU AGE® models for reversing and implementing in Java EE or .NET. It also includes the following use cases: product referencing and product ordering. Each TP requires a 2 man-day support and each batch process requires a 1 man-day support from each side (BLU AGE® expert group and STIME). Even though the manual building of models within BLU AGE® is strongly inspired by the success of the commented output of the reverse engineering process; a significant gap in research and development remains. One must indeed obtain maximum automation. Due to the ongoing nature of this experimentation, further details and insights will be provided when orally presenting this paper.

8. Perspectives, generalizing the approach

The reverse engineering process above described is strongly coupled with a given tool: VAP. Another modernization context associated with another legacy development tool or language requires other metamodels and other transformation chains. In short, the ultimate intention of the BLU AGE® REVERSE module is to become free from a legacy technology. To avoid the redefinition of such metamodels and transformation chains, a more generic framework is expected (Figure 12).

About transformation chains. The complexity of certain model transformations leads to unintelligible transformation programs. The need for segmentation of these complex transformations amounts to creating transformation chains. Transformation chains involve a special kind of models: weaving models. Instead of having sizeable metamodels which capture an entire domain, relationships and correspondences between the considered models could be described by specialized weaving models. Weaving models in essence depend upon the source and the target models of a transformation. This dependence is expressed in terms of processed model elements from the source, from the target and from markers in profiles.

In the transformation workflow from Figure 12, VAP may be viewed as a technical ontology: a set of technical concepts, terms (*Table*, *Column*… see prior sections) and their semantic relationships. These concepts are currently (manually) implemented by means of an Ecore metamodel. This means that, facing another legacy technology, a new metamodel of this technology has to be constructed. To eliminate, as much as possible, such tasks, a meta-metamodel is needed for instantiating a technology as a metamodel which conforms to this meta-metamodel. Besides, to create some interoperability not only within BLU AGE® but with other reverse engineering tools, a normative meta-metamodel is required. KDM may opportunely play this role.

The primary purpose of an enhanced reverse engineering process is thus the discovering of the technology itself in order to, interactively, populate the metamodel describing this technology. A knowledge base of the legacy technology is then produced which acts as a referent. The interactive aspect corresponds to the fact that manual interventions are necessary to classify all concepts of the targeted technology.

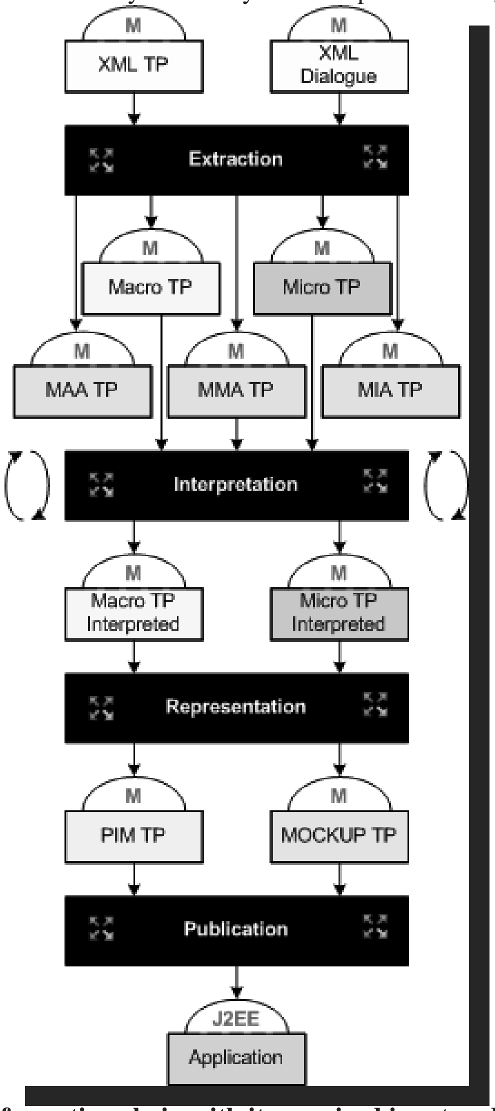


Figure 12. Transformation chain with its required input and output models

At this time, KDM is not supported but the transformation workflow in Figure 12 has been prepared to move to KDM. The macro/micro dichotomy is indeed a key notion in KDM and is linked to the *Micro KDM domain* [2]. This package includes the model elements which allow the characterization of elementary actions in the target legacy technology. For instance, a *MOVE* statement in COBOL which is also present in the VAP dialect maps to an instance of *ASSIGN* in KDM, a micro action.

The transformation chain in Figure 12 shows as outputs of the *Extraction* phase[12], five models[13]. The *Micro TP* model records all the concepts of the VAP programming language to manage a TP within the reverse engineering process. In the same logic, the *Macro TP* addresses small-scale architectural issues like, for instance, function keys in old character terminals which are used for screen linking.

The *Micro TP* and *Macro TP* models are thus two sets of VAP markers. Ideally, according to the KDM philosophy, they come from the extraction process. In our experiment (see prior sections), they have been constructed from scratch. In any case, they are independent of the application being reversed.

In contrast, the *MMA TP* model standing for *MAcro Annotated TP* and the *MIA TP* model standing for *MIcro Annotated TP* refer to all of the data capturing the nature and the status of a given TP. These two types of models are business-oriented. For example, one may find in a *MAA TP* model how display screens are linked, more precisely, which function key of the TP of interest leads to another TP. In a *MIA TP*, one may find, in a structured way, all the material found when parsing the code relating to a TP. By definition, the *MMA TP* and *MIA TP* models are annotated with markers coming from the *Micro TP* and *Macro TP* models but annotations are physically present in the weaving model: the MMA TP model (MMA stands for *Merged Model with Annotations*).

Annotations are twofold. First, as much as possible, the *MMA TP* and *MIA TP* models are automatically annotated. Next, business-oriented marks are appended by experts. Functional experts (requirements engineers) annotate extracted elements to make emergent and visible the application's functionalities while, by complementary, the technical material is distinguished from functionalities. The VAP profile (*Micro TP + Macro TP*) used to annotate more or less automatically the extracted models, is thus subject to increasing. This results from the need of contextual markers which refer to the discovered technology or the specific nature of the application: business domain, company, end-users…

The more interesting point is the availability of the *MMA TP* model, a weaving model. This model is not self-contained in the sense that it depends upon *MMA TP* and *MIA TP*. It is the global view of the reversed system since it formalizes and maintains the consistency between the macro and micro views promoted by KDM.

As shown in Figure 12, the interpretation phase is iterative. Several loops enable the identification, analysis and classification of the reversed material. For instance, (a) the marking of dead code[14], (b) the delimitation of "system code" like a CICS command, are important actions within the interpretation phase. The latter (b) can be either

[12] The *XML TP* and *XML Dialogue* input models as just results of filter application (*e.g.*, control character cleaning) on VAP rough files.

[13] A *M* mark on the top of a rectangle means "model".

[14] For example, specific database accesses which have no counterpart in the modern application, international date formatting which is already available the Java core library…

automatically detected with relevant information in the technology model or it is captured on-the-fly when no pre-existing knowledge exists.

This phase is strongly based on knowledge capitalization, *i.e.*, refactoring in order to determine what are the common (currently duplicated) parts of the application, the creation of reverse strategies like viewing a *GOTO* VAP command as a Java exception for instance, etc.

The representation aims at producing models in the BLU AGE® formalism (Figure 10). The current scheme in Figure 12 shows a TP-per-TP approach but a consolidation is absolutely necessary: each TP is a small piece of business functionality in the legacy application but only the merging of all TP and batch processes is the expected tangible result for end-users.

At this time, the representation phase generates the BLU AGE® PIM for a given TP only. The MOCKUP PIM is a mixing of XHTML code and BLU AGE® markers to make screens concrete in the new technology, here Web pages.

Another feature is the fact that the BLU AGE® models are also marked by means of VAP markers to create some traceability in the reverse engineering process. Moreover, these traces are intended to be used for assembling the different PIM/MOCKUP models resulting from different TP.

As for the publication phase, it is the ordinary BLU AGE® application generation process.

9. Conclusion

MDD is a promising technology to build large-scale applications from scratch, but the challenge of transforming legacy code into models stumbles over the heterogeneity of old systems. In this paper, the proposed approach benefits from the preexisting structuring of COBOL code and some associated environmental information (configuration parameters…) based on VisualAge PACBASE (VAP). Starting from an extraction module (PAF) of VAP, several layered metamodels and models are instantiated in sequence. The paper describes a case study that serves as a benchmark for perfecting a reverse engineering method and its implementation in the REVERSE module of the BLU AGE® CASE tool. At this time, several TP of the SCAFRUITS application have been redesigned and deployed in the Java EE platform. However, there is still no attempt about the assembling of these rebuilt TP to obtain a global renewed business application.

A key research focus is the systematic discovering of the legacy technology itself. The described project (as a perspective) is to concomitantly create knowledge about the reversed application and its support technology. This project is currently greatly inspired by the KDM standard. A transformation framework under specification and development is on purpose offered.

10. Bibliography

1. R. France, S. Ghosh, T. Dinh-Trong and A. Solberg: Model-Driven Development Using UML 2.0: Promises and Pitfalls, *IEEE Computer*, 39(2), pp. 59-66 (2006)
2. Architecture-Driven Modernization (ADM)/Knowledge Discovery Meta-Model (KDM), version 1.1, January 2009
3. S. Eveillard and A. Henry, *Transforming VAP metamodels into UML*, talk at the industrial track of MODELS'2008, Toulouse, France (2008)

Participation of Business Stakeholders in MDD

Frank Wille, Novulo, Auke Vleerstraat 6, Enschede, The Netherlands

Abstract. The trend of creating higher abstraction levels in software development has moved to the next step with Model Driven Development. Higher productivity, better quality, and easier maintenance are already well-known advantages, as well as simplifying the communication process. However, do our model presentations foster effective communication with the business users? Can we be certain that presenting a UML diagram or a data model to business stakeholders will allow them to confirm that it fits their specific needs?
At Novulo, we believe that the gap between business and IT is not bridged by MDD in its current form. Rather, by providing a no-nonsense and fully graphical presentation of UI, conditions and workflows in the model, and by facilitating the dependencies of data modeling and application modeling, we have created a new means of communicating productively with business stakeholders in the modeling phase. written.

Model-driven software development of distributed heterogeneous systems

Alexander Broekhuis (luminis) and Jeroen Kouwer (Thales)
Thales-luminis, The Netherlands

Thales radar systems are highly complex and demanding products. These distributed real-time systems are modeled on heterogeneous platforms with multiple technologies (languages, RTOSes and protocols). A typical radar system consists of more than 50 boards and processes data at a rate over 140 Mbytes per second.

This presentation is aimed on sharing the experiences of a model-driven approach to the software development of radar systems. Technical experiences on defining correct meta-models, model validation through the use of OCL and writing proprietary transformation rules. Important success factors in introducing MDA in the Thales software development process will be discussed. Such as the use of modeling guidelines, first level support and experience exchange meetings.

Within Thales the first step of introducing software development based on MDA is aimed on: the modeling of system structure, data communication between distributed software components and the mapping of software on specific hardware components. Using an Eclipse based tool chain UML models are validated through the use of OCL and transformed to XML. Finally code is generated in the form of component specific interfaces and technology specific optimized code for the target hardware platforms.

Used tools and technologies: openArchitectureWare, Eclipse Modeling Framework, UML 2, QVT, OCL. The audience is expected to be familiar with: UML, XSD/XML and MDA principles in general

Alexander Broekhuis has a background as an experienced software engineer, designing object-oriented software for high-impact industrial systems. Over the last years, Alexander has been involved in the introduction of MDA patterns and technologies in multi platform environments. This with a strong focus on meta-model definitions, model2model transformations and MDA tool chains.

Jeroen Kouwer is working as an architecture engineer within Thales Surface Radar. He has been working specifically on the introduction of UML modeling and automatic code generation for Thales radar architectures. This strongly focussing on high-throughput data communication and deployment on heterogeneous hardware and software platforms.

traceMAINTAINER – Tool Demonstration

Patrick Mäder[1], Orlena Gotel[2], and Ilka Philippow[1]

[1] Department of Software Systems
Ilmenau Technical University, Germany
patrick.maeder|ilka.philippow@tu-ilmenau.de
[2] Department of Computer Science
Pace University, New York, USA
ogotel@pace.edu

Abstract. This paper outlines a proposed demonstration that will present the core functionality of *traceMAINTAINER*, a prototype tool that enables the semi-automated maintenance of traceability relations held between different models of software systems expressed in UML. The demonstration will cover: how *traceMAINTAINER* integrates with UML modeling tools, such as Enterprise Architect; how traceability updates are handled based upon the recognition of development activities; what happens when decisions about a traceability update cannot be made fully automatically; and how to tailor the rules that guide the traceability maintenance process.

1 Introduction

traceMAINTAINER is a prototype tool that enables the semi-automated maintenance of traceability relations held between different models of software systems expressed in UML ([1], [2]). The approach analyses changes undertaken to structural UML models while working within a CASE tool that supports UML modeling. Change events are captured and sequences of events are sought that correspond to predefined rules. These rules represent the various ways in which recurring development activities can be undertaken and directives to update the impacted traceability relations once identified. This paper is provided as a companion to the architectural description of *traceMAINTAINER* provided in [3].

2 Integration with Case Tools

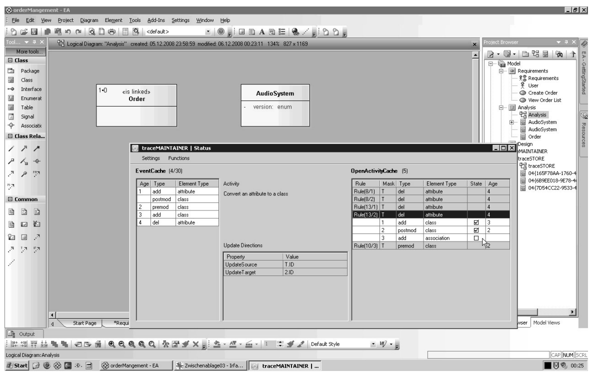
traceMAINTAINER is designed to work with traditional UML modeling tools to expand their support for traceability. The demonstration will begin by illustrating *traceMAINTAINER*'s integration with Enterprise Architect and by explaining how such integration can be extended to other UML modeling tools. The integration relies upon two specific components: an event generator that captures changes to supported model elements and a *traceSTORE* that extends the traceability functionality of the base CASE tool. *traceSTORE* holds traceability relations in an additional class model within the development model and we will discuss the benefits of this realization over the vanilla traceability functionality (of Enterprise Architect in this case). We will then create a small UML model for demonstration purposes. This will involve creating a use case model with two use cases, relating these to elements of a design model, and demonstrating *traceSTORE*'s ability to show and navigate the traceability relations in both models. We will also emphasize the role of the indicators that we have introduced to recognize the existing traceability relations on an element and their count. We will further explain the structure of the element references and relations while creating the example model.

3 Recognizing Development Activities that Impact Traceability

In this part of the demonstration, we will show how *traceMAINTAINER* is used in the context of software development. Expanding the above example, we will show how an analysis model may need to be changed based upon a change request relating to a use case. We will use the development activity of extracting an attribute into its own class to discuss the functionality resulting from this change request and the use of *traceMAINTAINER* to account for it. We will then provide a step-by-step walkthrough as to the tasks undertaken and the response of *traceMAINTAINER* to automatically maintain the traceability. During these activities, we will show the rule engine status window (see Figure 1) to explain how the development activities are recognized by *traceMAINTAINER*. We will also explain how the traceability relations are updated after recognizing the development activity and the kind of feedback the user receives.

4 Semi-automated Traceability Maintenance

In the preceding parts of the demonstration, we will have presented the use of *traceMAINTAINER* in a context were no user interaction was necessary (automated traceability maintenance). Nevertheless, while development activities can be recognized by *traceMAINTAINER*, it is not always possible to make a definitive update of traceability relations. In the next part of the demonstration, we will present an example according to Figure 2 in which user interaction is required. After recognizing the development activity, *traceMAINTAINER* presents the user with a dialog that lists all the existing and potentially new traceability relations involved in the activity. The user is required to decide how to handle certain relations. We will explain when there is the necessity for user interaction, as well as the reason for the different alternatives presented. After choosing the preferred alternative, we will show how the traceability relations then get updated (as per the previous section).



traceMAINTAINER | Status

EventCache (4/30)

Age	Type	Element Type
1	add	attribute
	postmod	class
2	premod	class
3	add	class
4	del	attribute

Activity

Convert an attribute to a class

Update Directions

Property	Value
UpdateSource	T.ID
UpdateTarget	2.ID

OpenActivityCache (5)

Rule	Mask	Type	Element Type	State	Age
Rule(8/1)	T	del	attribute		4
Rule(8/2)	T	del	attribute		4
Rule(13/1)	T	del	attribute		4
Rule(13/2)	T	del	attribute		4
	1	add	class	✓	3
	2	postmod	class	✓	2
	3	add	association	☐	
Rule(10/3)	T	premod	class		2

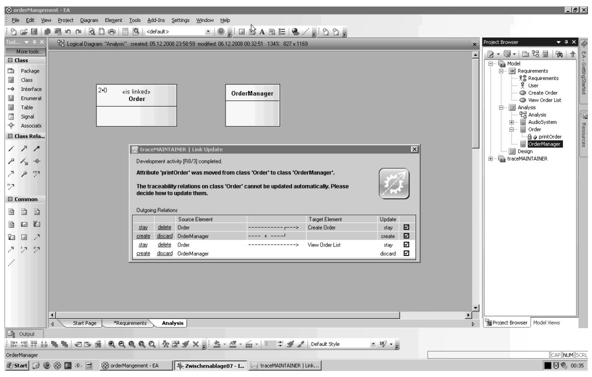
Fig. 1. Before performing the last change of the development activity, the *traceMAINTAINER* status window shows that the correct rule has been instantiated as an OpenActivity and that all already performed changes have been correctly assigned to it. The remaining mask is comparable and requires creating an association between both classes involved in the development activity.

5 Changing Traceability Maintenance Rules

The current rule catalog used by *traceMAINTAINER* has been in use for about one year in different experiments and by industry partners. During this period it has been stable. However, it is unlikely that this set of rules is fully complete and correct. Furthermore, there are development activities that may be carried out in different ways depending on the domain of the project and the pre-disposition of the developers. It is therefore desirable to be able to customize the existing rules and to support the traceability of new model elements. In this part of the demonstration, we will present the underlying rule catalog of *traceMAINTAINER*. We will describe how these rules were derived and function. We will also illustrate how they are stored and describe their internal structure. We will further create a new rule and, using this example, discuss the concept of properties, masks, alternatives and traceability updates (all key concepts underlying the traceability maintenance process supported by *traceMAINTAINER*). We will demonstrate different ways to define properties and explain how they help in creating *good* rules. Finally, we will highlight the process for validating new or changed rules.

6 Status

traceMAINTAINER provides an extensive set of features for maintaining traceability between a broad spectrum of UML model element types. Results show that the approach is capable of reducing the effort (and so the cost) of maintaining traceability quite dramatically and at quality levels comparable to manual



Development activity [R8/3] completed.

Attribute 'printOrder' was moved from class 'Order' to class 'OrderManager'.

The traceability relations on class 'Order' cannot be updated automatically. Please decide how to update them.

Outgoing Relations

		Source Element		Target Element	Update	
stay	delete	Order	------------┌--->	Create Order	stay	☑
create	discard	OrderManager	---- + ----┘		create	☑
stay	delete	Order	--------------->	View Order List	stay	☑
create	discard	OrderManager			discard	☑

Fig. 2. *traceMAINTAINER* dialog that informs the user about a recognized development activity and requires a decision to be made between several options in those cases where the traceability maintenance cannot be determined automatically.

maintenance. The approach is intended as a complement to those approaches that initially create traceability relations using either manual or automated techniques. The reader is directed to a companion publication for a detailed architectural description on *traceMAINTAINER* [3].

Acknowledgments The authors would like to thank Tobias Kuschke, Christian Kittler and Arne Roßmanith for implementing the *traceMAINTAINER* prototype.

References

1. Mäder, P., Gotel, O., Philippow, I.: Rule-based maintenance of post-requirements traceability relations. In: Proc. 16th Int'l Requirements Eng. Conf., Barcelona, Spain (September 2008)
2. Mäder, P., Gotel, O., Philippow, I.: Enabling automated traceability maintenance by recognizing development activities applied to models. In: Proc. 23rd Int'l Conf. on Automated Software Engineering ASE, L'Aquila, Italy (September 2008)
3. Mäder, P., Gotel, O., Philippow, I.: traceMaintainer: A Tool for the Semi-automated Maintenance of Model Traceability. In: (submitted), Enschede, Netherlands (June 2009)

SOA and SHA Tools Developed in SHAPE Project

Andrey Sadovykh[1], Christian Hahn[2], Dima Panfilenko[3], Omair Shafiq[4], Andreas Limyr[5],

[1] SOFTEAM, Paris, France
andrey.sadovykh@softeam.fr
[2] DFKI MAS, Saarbrücken, Germany
christian.hahn@dfki.de
[3] DFKI IWI, Saarbrücken, Germany
dima.panfilenko@iwi.dfki.de
[4] STI, Innsbruck, Austria
omair.shafiq@sti2.at
[5] SINTEF, Oslo, Norway
andreas.limyr@sintef.no

Abstract. This article presents the SHAPE project tool set dedicated to Model Driven Engineering (MDE) methodology for Service Oriented Architectures (SOA) and Semantically-enabled Heterogeneous service Architectures (SHA)

Keywords: SOA, SHA, SoaML, SHAPE, Web Services, Agents, CIM, PIM, PSM.

1 Introduction

The SHAPE FP7 project [1] aims at the following goal:

To specify, develop and test a tool supported methodology for designing implementing flexible business models and parameterized services on a Semantically-enabled Heterogenious service Architecture (SHA) through model driven engineering (MDE) approaches and standardization of these results.

In this context the SHAPE relies on the OMG SoaML specification and intends to extend it for heterogeneous service architectures.

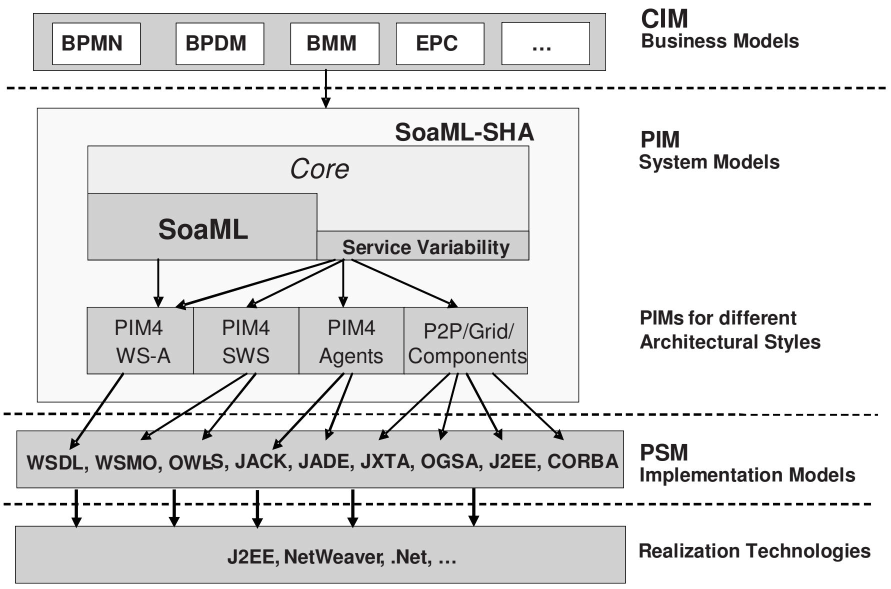


Fig 1 SHAPE Concept Overview

The SHAPE tool set has to cover all MDA layers for SOA and SHA as presented in Fig. 1.

The implementation of the SHAPE tool set is presented in details below.

2 SHAPE Tool Set

The project intends to integrate various MDE tools in the field of flexible business models, Enterprise Architectures, SOA Semantical Web Services and Agents.

The SHAPE partners develop the following components:

- Flexible business models – CIMFlex by DFKI IWI and Objecteering Enterprise Architect by SOFTEAM
- System models – Objecteering Enterprise Architect and UML Designer by SOFTEAM, PIM4Agents by DFKI MAS, Composition Studio by SINTEF
- Implementation Models – Objecteering Java Developer and Web Services Developer by SOFTEAM, semantic Web Services Modelling Toolkit by STI

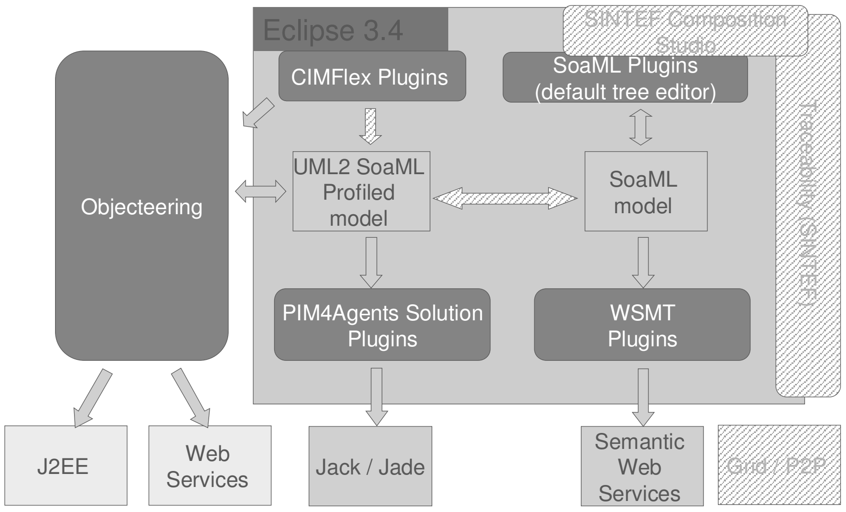


Fig 2 SHAPE Tool Set Components

Fig. 2 depicts the current component architecture for the SHAPE tool set.

The tool set is integrated using Eclipse EMF technologies over SoaML models export/import, while the individual aspects of SOA or SHA are treated by corresponding tools.

The details on each component are provided in the sections below.

2.2 Objecteering CASE Tool

Description:

In the context of the SHAPE project SOFTEAM proposes Objecteering 6 CASE Tool [2]. It provides a complete, simple to use model-driven development solution, dedicated to expressing and managing requirements, building complete and accurate UML, Enterprise Architecture and Business models, generating a full range of documentation and automating application code production for Java/EJB, C++, C#/.Net, SQL, CORBA and Fortran. With more than 250 interactive real-time consistency checks, Objecteering 6 manages model consistency in order to guarantee high quality models and correct code generation. Live traceability links are managed throughout the entire development cycle, from requirements, analysis and design through code generation, tests and application deployment.

In addition, Objecteering 6 provides teamwork facilities through a multi-user repository and flexibly supports cooperative work, with no limits regarding large-scale developments. In order to allow concurrent modeling, a lock mechanism can be applied down to class level, thereby guaranteeing the consistency of the model shared by team members. Branches are managed through the model diff/merge function. A powerful model component feature can be used to organize project development over several different teams. Once packaged, model components can be easily deployed to efficiently manage communication and model delivery between the different teams involved in a project.

The Objecteering MDA Modeler automates the OMG's MDA approach, using standardized or specialized UML profiles. Objecteering MDA Modeler is used to define dedicated tools for code generation, to support specific target platforms, and to support methodologies and modeling processes specific to each organization.

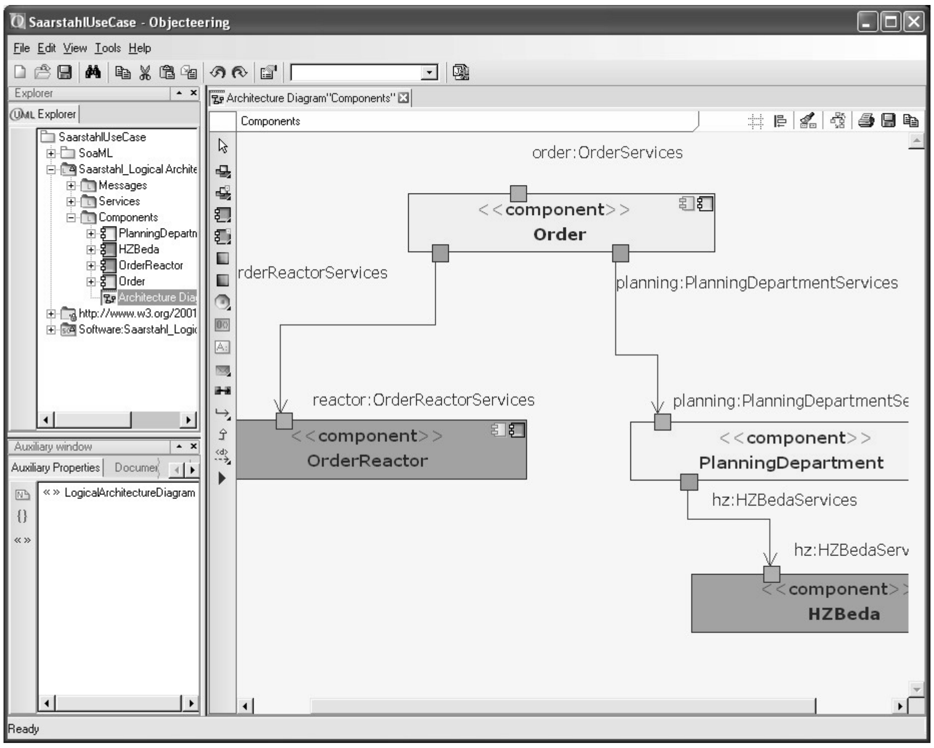


Fig 3 Objecteering Enterprise Architect - Logical Architecture View

Role in SHAPE:

In the frame of the SHAPE project, SOFTEAM implements a dedicated tool support for SoaML and ShaML. These specifications are implemented as UML2 Profiles in the Objecteering CASE Tool. In addition the SOFTEAM works on integration of the SoaML to its Enterprise Architecture and Business modeling solution.

The Objecteering tool is used main means to edit and visualize the SoaML models. The SoaML models are then exported in order to make them available for the other tools presented below.

In the context of the industrial use cases the Objecteering is used for the forward MDE in the SOA field. The business models created in BMM are translated in Enterprise Architecture models in order to finally obtain the models for conventional Web Services and Java platform. The Java code and conventional Web Services are automatically generated.

2.3 CIMFlex

Description:

The CIMFlex Editor allows the user to create and refine a semi-formal model of a business process, an organisational structure, a data structure or business rules based on the input coming from the domain users (see Fig 4). The editor is able to create, change and store these types of models in EPC or BPMN notation. As storage format XML files are generated. The target users of this component are the domain user and especially the business analysts.

From an architectural point of view the component has two interdependencies with other components for its output. The information, which is required for the creation of a CIM model, will be derived from the use cases by the domain users.

The output of the CIM level editor can have two different forms depending on its purpose. On the one hand, a model on CIM level in BPMN notation can be used as the technical information description draft, giving a starting point for the transformation into BPEL for further execution of the resulting model or the enrichment with further technical information.

On the other hand the output of the CIM level editor is the starting point for the CIM to PIM transformation. In this case the editor doesn't provide the models in BPMN notation, but transforms them into SoaML models. This functionality is now under development on a conceptual level and should be implemented as soon as the initial version of a transformation is agreed upon.

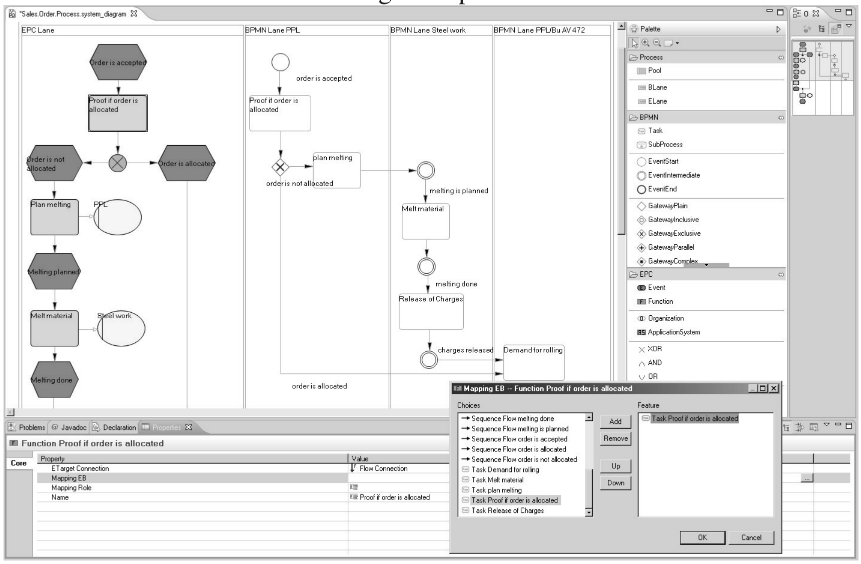


Fig 4 CIMFlex process editor

Role in SHAPE:

One of the main usages of the CIMFlex tool and its underlying metamodel is description and deployment of services and services architectures at the CIM-level to support business scenarios. On the other hand, SHAPE project promotes the usage of MDA idea to improve the development process and SoaML idea for the description of the services. In order to enable Flexible Business Models to accomplish this, it is necessary to align them with the SoaML. The CIMFlex language is needed to integrate domain experts which are involved in a business process and their knowledge into the software development. It has to be a trade off between the "non-IT oriented" people who are the target users of a new system and the people who implement it.

The CIMFlex tool should provide a means for aiding domain experts with the documentation of their knowledge on the CIM-level in a set of models including data, organisational, business rules and process diagrams. Provided that, these CIM-level models should be then transformed into the PIM-level models, thus transferring the domain expert knowledge on the technical level, allowing for further enrichment and amendments. At the same time, the models in BPMN notation can serve as a starting point for the further transformation to BPEL in order to provide the input for the execution engines.

The component was implemented at the Institute for Information Systems at the German Research Center for Artificial Intelligence (DFKI IWi[1]).

2.4 PIM4Agents

Description:
For designing multiagent systems (MASs), DFKI developed a platform-independent domain specific modeling language for MAS called Dsml4MAS [7] in accordance to the language-driven initiative. Like any other language, Dsml4mas consists of an abstract syntax, formal semantics, and concrete syntax:

- The abstract syntax of Dsml4MAS is defined by a platform independent metamodel for MAS called PIM4Agents defining the vocabulary in terms of concepts and their relationships
- The formal dynamic and static semantics is expressed using the specification language Object-Z which is a stated-based and object-oriented specification language.
- The concrete syntax is defined as set of notations facilitating the presentation and construction of Dsml4MAS. It is specified using the Graphical Modeling Framework (GMF) that provides the fundamental infrastructure and components for developing visual design and modeling surfaces in Eclipse.

The corresponding Dsml4MAS Development Environment[2] (DDE) allows us to specify MASs with Dsml4MAS and to transform the created models to agent execution platforms like Jack Intelligent Agents and Jade (Java Agent Development Framework).

Role in SHAPE:
The role of Dsml4MAS and PIM4Agents in particular is to serve as intermediate level that is already enriched with platform specific information on agent-related aspects.

[1] http://iwi.dfki.de
[2] http://dsml4mas.sourceforge.net

Consequently, the model transformations to Jack and Jade can be used to generate executable code based on the SoaML description.

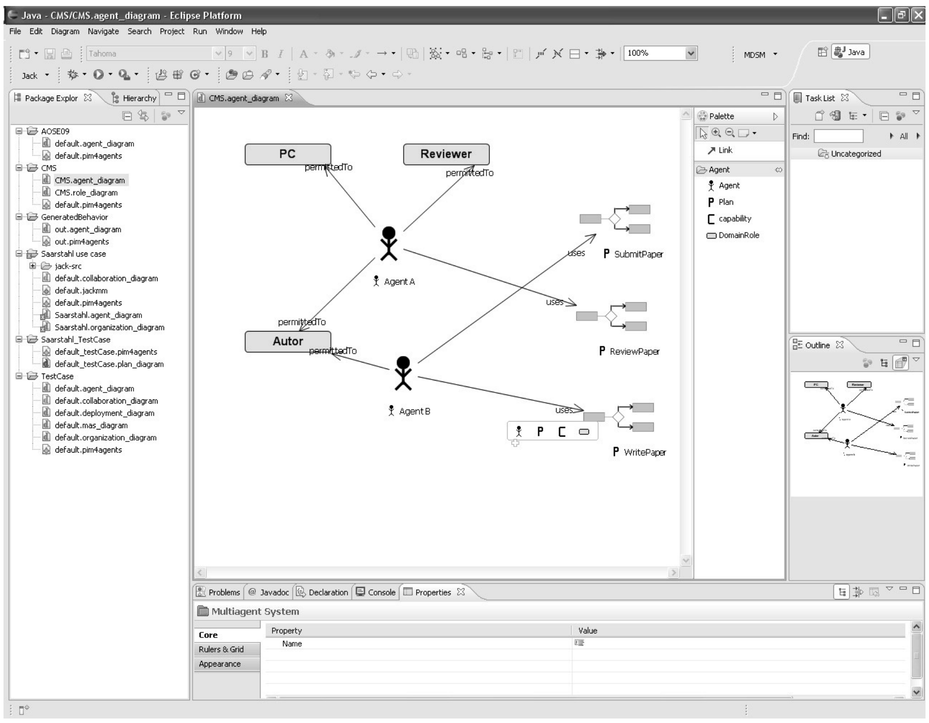


Fig 5 The PIM4Agents Development Environment

Through the model transformation from SoaML to PIM4Agents and the formal semantics of PIM4Agents, SoaML gets a clear formal semantics that can be used for reasoning purposes.

Moreover, particular concept from the PIM4Agents metamodel could be lifted to make SoaML more comprehensive with respect to modeling agent systems which might be necessary for complex scenarios like the Saarstahl use case that will be introduced in Section 3.1.

2.5 Web Services Modeling Toolkit

Description:

The Web Services Modeling Toolkit (WSMT)[3] [7] is an Integrated Development Environment (IDE) for Semantic Web Services implemented in the Eclipse framework. WSMT aims to aid developers of Semantic Web Services through the WSMO paradigm, by providing a seamless set of tools to improve their productivity. As already mentioned an Integrated Development Environment (IDE) is defined as a

[3] http://sourceforge.net/projects/wsmt

type of computer software that assists computer programmers to develop software. IDE's like the Eclipse Java Development Toolkit (JDT) and NetBeans for developing java software have proven that good tool support can improve the productivity of engineers. It could be said that the time of ontology and Semantic Web Service engineers is a more precious commodity, as the number of people who are currently skilled in conceptual modeling is much less than those that can code in Java. This underscores the need for adequate tool support for working with semantic technologies and an IDE can tie together these tools in such a way that the whole application is more than the sum of the parts that make it up. WSMT facilitates users in validating the semantic description of services in WSML, text editor as well as visualizing the ontologies and service descriptions.

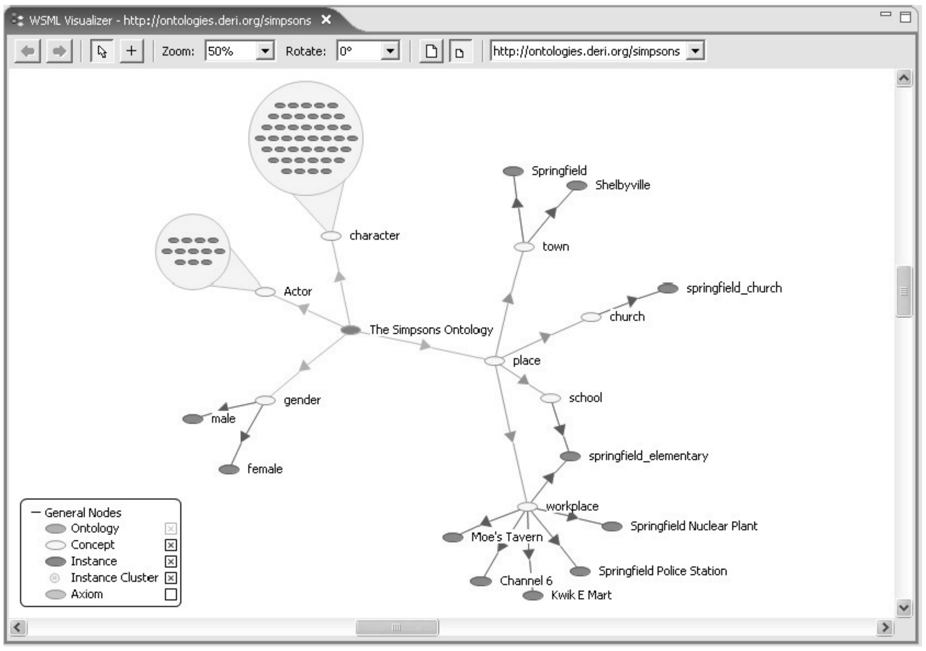


Fig 6 Ontology in WSML Visualizer

Role in SHAPE:

Role of WSMT in SHAPE MDE Toolkit is to validate and show the generated WSML code from SoaML, using text-based editor as well as visualizer. The WSMT further provides support to engineers in developing further the generated Semantic Web Service descriptions using text-based editors and visualizers of WSMT.

WSML Validation is supported in WSMT through an object model (WSMO4J) for manipulating WSMO descriptions and is capable of parsing and serializing WSML documents to and from this object model. It also provides a validator for WSML variants, which is exploited within the WSMT to validate the files that are located within the WSMT workspace as the user edits these documents. This validation ensures that the engineer of the semantic descriptions gets immediate feedback of errors they create, both in the syntax and the semantics of the semantic descriptions, as they create them.

Creation of Semantic Web Services descriptions by hand is also supported by WSMT through text-editor. It is very tempting when creating a toolkit to abstract away from a text editor and to provide more advanced editing support; However in many cases the engineer is more comfortable with editing the raw text of the semantic description. This is especially true with respect to WSML, as the WSML human readable syntax is a very lightweight syntax. WSMT caters for such users and provide them with additional features including syntax highlighting, syntax completion, in line error notification, content folding and bracket highlighting.

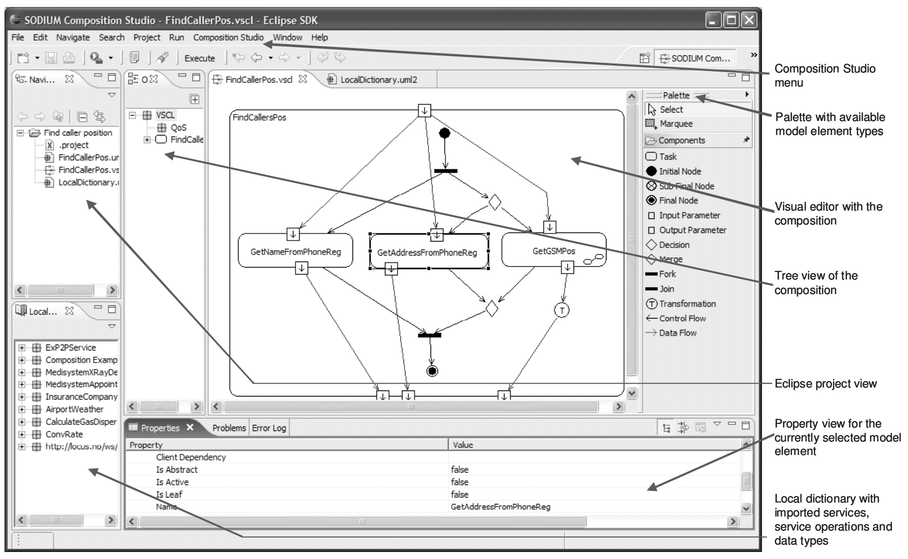
WSML based Semantic Web Services description code can also be visualized using WSMO visualizer provided in WSMT. The visualizer allows engineers to learn more information about their semantic descriptions as they create them. Normally visualization solutions are bolted on top of existing ontology engineering solutions after the fact, by providing an integrated solution the user need not switch back and forth between an editing environment and a visualizer to understand the effects of changes to the ontology.

2.6 Composition Studio

Description:

Service Oriented Architecture and Semantic Web Services have emerged as solutions for achieving interoperability between computer systems. Unfortunately, these fields comprise a large set of languages and standards, and intimate knowledge of these languages and standards is often needed when developing web services. Hence, considerable training time is required to educate service developers. There is also a lack of tools that support the development process so that the cost of development is high.

Composition Studio is a general toolkit for composition of semantic web services, and uses OMG's Model-Driven Architecture (MDA) [8,9] approach to software development. MDA separates the application logic from the execution platform by using models and transformations. It enables the user to specify the software with platform independent models that hide the details of the execution platform. The platform independent models can then be transformed automatically into platform specific models that include the details of the execution platform. Knowledge of the technical details of the execution platform is thus not needed with this methodology. However, the platform independent model must be defined in a language. For this purpose, the Composition Studio uses a variant of UML Activity Diagrams. UML has become an important standard for software engineering, and many software engineers are familiar with this standard. By using MDA and UML, the Composition Studio should be able to hide some of the complexity of the web service technology and thus reduce the entry level of building web service compositions.



Property	Value
Client Dependency	
Is Abstract	false
Is Active	false
Is Leaf	false
Name	GetAddressFromPhoneReg

Fig 7 Composition Studio

Role in SHAPE:

Composition Studio will be seen in relation to SoaML and how it can be integrated with the rest of the SHAPE toolset. Integration will mainly be performed by creating a set of transformations to and from the Composition Studio so that the results of other tools can be used in the Composition Studio and so that the result of the Composition Studio can be used as input to other tools.

Composition Studio creates models of service compositions with help of a visual model-based editor based on UML activity diagrams. The goal of this task is to be able to generate executable code that can be deployed on an execution platform. We will specifically look into BPEL as the execution platform.

The relation with high-level business modeling will also be investigated. Activity diagrams have much in common with many of the business process languages that exist today. Maybe modeling business processes in a service oriented context can give some advantages when it comes to generating code from such a high level of abstraction.

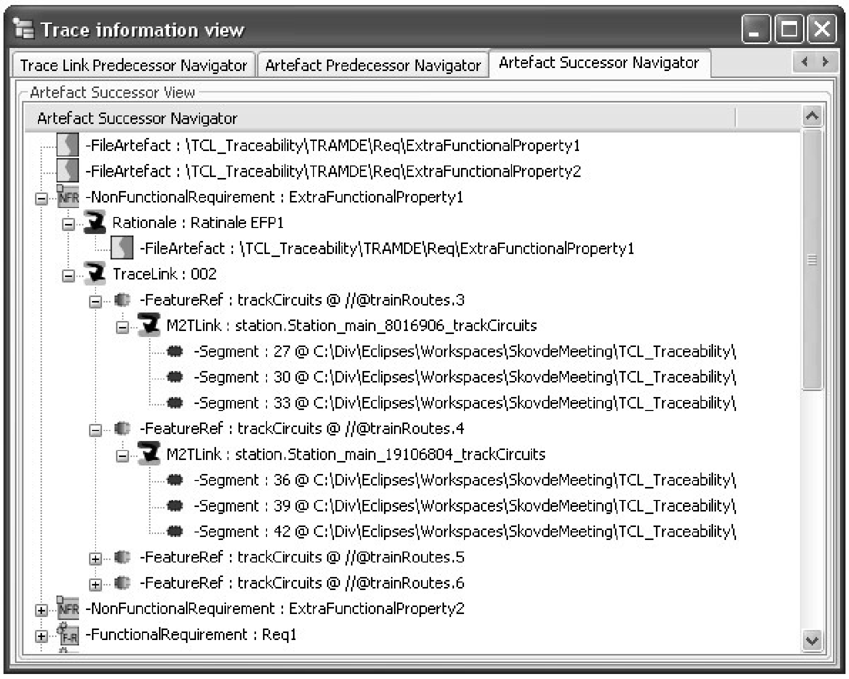
2.7 Traceability Tool by SINTEF

Description:

One of the main challenges in MDD is the management of relations between different artefacts produced in the development process. As systems become more complex, the number of artefacts is increasing. Furthermore, the artefacts are often generated. Therefore, trace links are needed to fully understand the many dependencies that exist between the different artefacts.

The reason to create and update traceability links is that the links can be used to support and document the development process. The information can be used in several ways, but the most obvious scenario is simple *trace inspection*. Through trace inspection it is possible to browse the trace information and get insight in how the different artefacts are connected. This is becoming more useful as an increasing number of artefacts are generated automatically from model to model and model to text transformations.

The TRAMDE Trace Analyser Tool is a prototype that implements the TRAMDE trace model. The tool is mainly focused on storing automatic traces produced by model transformation technologies like ATL and MOFScript. It also gives a view to the trace model and the possibility to analyse the trace model in different ways.


Trace information view
Trace Link Predecessor Navigator
Artefact Predecessor Navigator
Artefact Successor Navigator
Artefact Successor View

```
Artefact Successor Navigator
  -FileArtefact : \TCL_Traceability\TRAMDE\Req\ExtraFunctionalProperty1
  -FileArtefact : \TCL_Traceability\TRAMDE\Req\ExtraFunctionalProperty2
  -NonFunctionalRequirement : ExtraFunctionalProperty1
    Rationale : Ratinale EFP1
      -FileArtefact : \TCL_Traceability\TRAMDE\Req\ExtraFunctionalProperty1
    TraceLink : 002
      -FeatureRef : trackCircuits @ //@trainRoutes.3
        M2TLink : station.Station_main_8016906_trackCircuits
          -Segment : 27 @ C:\Div\Eclipses\Workspaces\SkovdeMeeting\TCL_Traceability\
          -Segment : 30 @ C:\Div\Eclipses\Workspaces\SkovdeMeeting\TCL_Traceability\
          -Segment : 33 @ C:\Div\Eclipses\Workspaces\SkovdeMeeting\TCL_Traceability\
      -FeatureRef : trackCircuits @ //@trainRoutes.4
        M2TLink : station.Station_main_19106804_trackCircuits
          -Segment : 36 @ C:\Div\Eclipses\Workspaces\SkovdeMeeting\TCL_Traceability\
          -Segment : 39 @ C:\Div\Eclipses\Workspaces\SkovdeMeeting\TCL_Traceability\
          -Segment : 42 @ C:\Div\Eclipses\Workspaces\SkovdeMeeting\TCL_Traceability\
      -FeatureRef : trackCircuits @ //@trainRoutes.5
      -FeatureRef : trackCircuits @ //@trainRoutes.6
  -NonFunctionalRequirement : ExtraFunctionalProperty2
  -FunctionalRequirement : Req1
```

Fig 8 TRAMDE Trace Analyser Tool

Role in SHAPE:

The TRAMDE Trace Model and Trace Analyser Tool will be used to create and analyse trace models generated by different transformations performed by tools in the SHAPE toolset.

It will be possible to get information on transformations to and from different tools. The trace model can be used to see if the transformation is sound.

Coverage analysis is useful for checking and ensuring that all relevant parts of the model are actually utilised by a transformation. If there are no traces from a particular model element, it is not used in the transformation.

Impact analysis in text transformation can allow for checking the impact of a model change to existing generated code. A limitation in this regard is unprotected areas in the code that use model references, which cannot be seen from the traceability model.

Orphans can occur in the code if model elements are deleted. There will then be traces from old model elements to the code. The transformation needs to be re-run in order to synchronise the model, the code, and the traces.

3 Use Cases Overview

The SHAPE methodlogy and tools are validated with two industrial use cases presented in the section below.

3.1 Saarstahl

Saarstahl AG, with its locations in Völklingen, Burbach and Neunkirchen along with Roheisengesellschaft Saar in Dillingen (Saarstahl and Dillinger Hütte each with 50%) is one of the most important manufacturers of long products in the world.

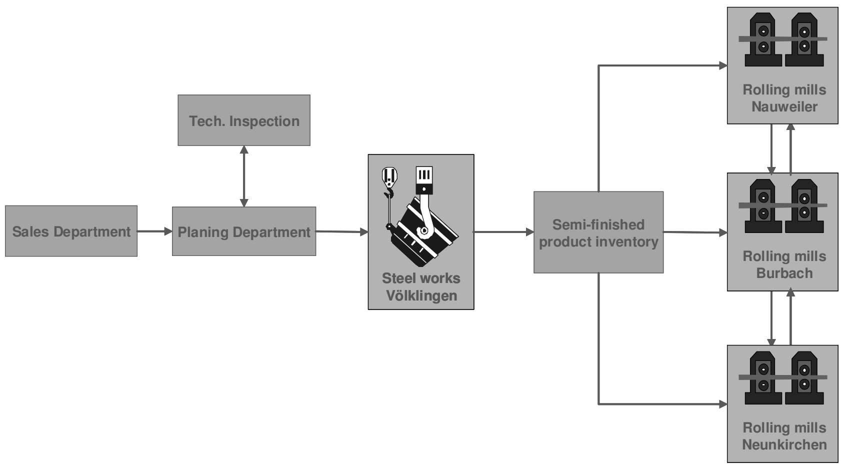


Fig 9 An abstract view on Saarstahl's supply chain

The Saarstahl case within SHAPE is a proof of concept for designing the main processes within the supply chain based on the results of SHAPE. Several challenges have to be addressed when it comes to a service-oriented design of the complete supply chain. From the viewpoint of Saarstahl it is fundamental that (i) business requirements that are specified by Saarstahl can easily be translated into a running system and (ii) existing systems (e.g. data bases) can be re-used within the SOA to maintain the high product quality. Both requirements match nicely with the SHAPE approach of a semantically-enabled heterogeneous service architecture as:

- SHAPE provides a full transformation path from the business level to adaptive platforms through MDE and thus allows Saarstahl to define executable artefacts in a very abstract manner on the CIM level.
- SHAPE supports the integration of existing legacy systems that are situated at different locations (e.g. Völklingen, Burbach and Neunkirchen) through a combination of Web Services and agents. A combination of adaptive execution platforms offers various advantages that allow to increase efficiency during run-time. The Saarstahl pilot consists of the following steps:
- Specification of business models and requirements: Saarstahl will formalize business models and requirements on the CIM level using CIMFlex. The business models will contain information with respect to involved organizational units, provided functionalities and exchanged data and resources.
- Model transformations from CIMFlex to SoaML, the generated business models serve as input.
- Model transformations from SoaML to Dsml4MAS and Web Services.

3.2 StaoilHydro

StatoilHydro is an integrated technology-based international energy company primarily focused on upstream oil and gas operations. A joint venture Production & Process Optimization project[4] between StatoilHydro and Schlumberger is aiming at improved reservoir management and optimize the reservoir performance over the life of the field.

In order to support the change management process, a prototyping process has been developed for surveillance workflow development and deployment. The dynamic prototyping process allows a bottom-up development of workflows starting from data conditioning and reconciliation up to optimization and control. This should work together with a workflow advisor enabled for improved decision making, streamlining of model updating and other tasks. The workflows depend on a set of software components as shown in the figure below.

In the SHAPE project we are investigating how to further improve this process by applying service-oriented technologies developed in SHAPE.

[4] http://www.spe.org/elibrary/servlet/spepreview?id=SPE-110655-MS&speCommonAppContext=ELIBRARY

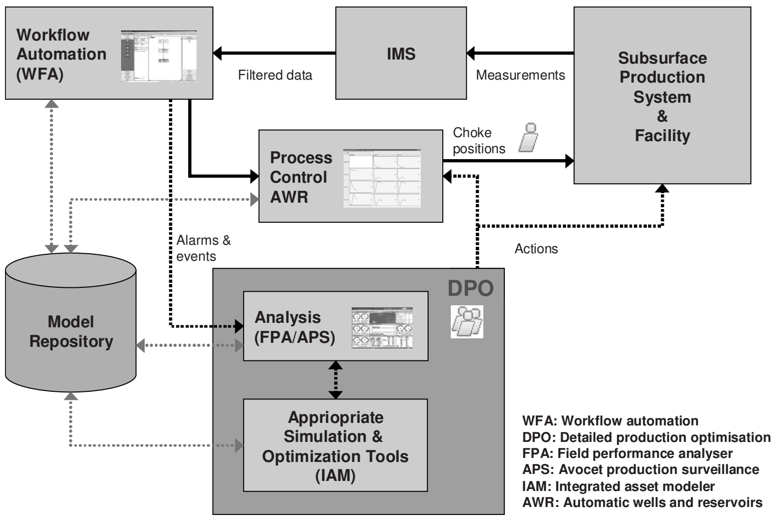
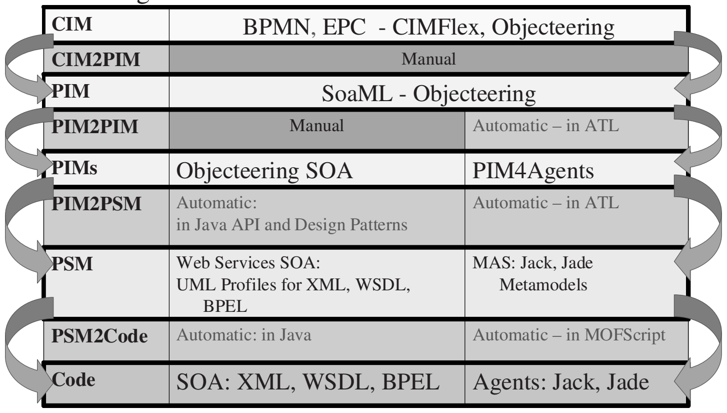

Workflow Automation (WFA)
IMS
Subsurface Production System & Facility
Filtered data
Measurements
Choke positions
Process Control AWR
Alarms & events
Actions
Model Repository
DPO
Analysis (FPA/APS)
Appriopriate Simulation & Optimization Tools (IAM)
WFA: Workflow automation
DPO: Detailed production optimisation
FPA: Field performance analyser
APS: Avocet production surveillance
IAM: Integrated asset modeler
AWR: Automatic wells and reservoirs

Fig 10 StaoilHydro Use Case

3 Progress and Future Plans

In the current stage all the tools were intentegrated in the single bundle, which will be available on the SHAPE's web site [1].

We experimented with the use cases demonstrated the full transformation flow presented in Fig. 11.



CIM	BPMN, EPC - CIMFlex, Objecteering	
CIM2PIM	Manual	
PIM	SoaML - Objecteering	
PIM2PIM	Manual	Automatic – in ATL
PIMs	Objecteering SOA	PIM4Agents
PIM2PSM	Automatic: in Java API and Design Patterns	Automatic – in ATL
PSM	Web Services SOA: UML Profiles for XML, WSDL, BPEL	MAS: Jack, Jade Metamodels
PSM2Code	Automatic: in Java	Automatic – in MOFScript
Code	SOA: XML, WSDL, BPEL	Agents: Jack, Jade

Fig 11 Currently supported transformation flow

In our future work we target implementation of the new coming ShaML specification that extends SoaML for semantic web services and agents. In addition the end-users highly require enhanced and seamless tools integration.

References

1. SHAPE project web-site, http://www.shape-project.eu/, visited on 20/04/2009
2. Objecteering web-site, http://www.objecteering.com, visited on 20/04/2009
3. SOFTEAM's R&D Department web-site, http://rd.softeam.com, visited on 20/04/2009
4. OMG web-site, http://www.omg.org, visited on 20/04/2009
5. OMG SoaML specification, http://www.omg.org/docs/ad/08-08-04.pdf, visited on 20/04/2009
6. Hahn, C.,Madrigal-Mora, C. and Fischer, K. (2008a). A platform-independent metamodel for multiagent systems, *International Journal on Autonomous Agents andMulti-AgentSystems*.
7. Mick Kerrigan, Adrian Mocan, Martin Tanler and Dieter Fensel: The Web Service Modeling Toolkit - An Integrated Development Environment for Semantic Web Services (System Description), Proceedings of the 4th European Semantic Web Conference (ESWC), June 2007, Innsbruck, Austria.
8. Kleppe, A., Warmer, J. and Bast., W., "MDA Explained, The Model-Driven Architecture: Practice and Promise" Addison Wesley, 2003
9. OMG, MDA Guide Version 1.0.1, J.M.a.J. Mukerji, Editor. 2003, OMG.

EMFText and JaMoPP - Tool Presentation

Florian Heidenreich, Jendrik Johannes, Sven Karol,
Mirko Seifert, and Christian Wende

Institut für Software- und Multimediatechnik
Technische Universität Dresden
D-01062, Dresden, Germany
{florian.heidenreich,jendrik.johannes,sven.karol,
mirko.seifert,c.wende}@tu-dresden.de

Abstract. Textual Syntax (TS) as a form of model representation has made its way to the Model-Driven Software Development community and is considered a viable alternative to graphical representations. In this demo we present EMFText [1], an EMF/Eclipse integrated tool for TS development, where we focus on its abilities to generate refineable default syntaxes and handling of large languages, e.g. Java., with EMFText.

1 EMFText - Tool Introduction

In the last few years, a variety of tools for mapping between models and concrete syntax have emerged to support the design and implementation of text editing facilities for modelling languages. The textual syntax built with these tools can be either generic or custom. Generic syntaxes like HUTN [2] are defined on the level of metamodelling languages. Thus, they are either instantly available or can be derived automatically for concrete modelling languages. Custom syntaxes need manual specification effort. The former are also often more verbose and do not directly reflect the semantics of a language, in contrast to the latter, which are more compact and use symbols that ease reading.

To bridge the gap between these two flavours of textual syntax, we developed EMFText, which allows to stepwise refine specifications that are automatically derived from given metamodels and, thus, enables the developer to build custom syntax starting from a generic syntax. To exemplify the broad applicability of our approach, we present both the derivation of generic syntax from a given metamodel and the development of a highly customised syntax in this demo.

The remainder of this demo paper is organised as follows: Section 1.1 gives an overview of the features of EMFText and Section 1.2 summarises some applications of EMFText. We then present two applications in more detail in Sections 2 and 3. For a detailed overview of EMFText, please refer to [3].

1.1 EMFText Features

EMFText supports text syntax development in the context of MDSD with a variety of features. Amongst other things, development, evolution and modularization of text syntax are well supported. The most outstanding features are:

Automatic generation of default syntax With EMFText, an initial syntax can be generated with a single click for any metamodel. The generation mechanism conforms to the HUTN standard and the generated specification can be further tailored towards specific needs.

Simple and precise syntax specification EMFText comes with a simple but rich syntax specification language called *ConcreteSyntax (CS)*. The language is based on EBNF and follows the concept of *convention over configuration*. This allows for very compact and intuitive syntax specifications, but still supports tweaking specifics where needed.

Modular specification EMFText has an import mechanism that not only supports specification of a single text syntax for multiple related Ecore models, but also allows for modularization and extension of CS specifications.

Default reference resolving mechanisms A default name resolution mechanism for models with globally unique names is available out of the box for any syntax. More complex resolution mechanisms can be realized by implementing generated resolving methods through which also inter-model references can be established.

Comprehensive syntax analysis A number of analyses of CS specifications inform the developer about potential errors in the syntax—like missing syntax for a certain metaclasses.

Generic editor with outline view and preference page The EMFText editor fully integrates with the Eclipse Platform. It supports the outline and preference pages concept of Eclipse using the EMF Edit facilities. Through this, icons and labels defined for a metamodel are reused and can be recognized in the text editor.

Customizable Syntax Highlighting The editor supports code highlighting that can be individually customized for each text syntax.

Code Completion Code completion is supported for references and can be easily tailored by customizing the reference resolving mechanism.

1.2 Examples for Textual Syntax

We have gained experience in using EMFText by defining textual syntax for several domain specific and general purpose languages.[1] The examples for EMFText-based syntax range from simple tree-shaped models like *feature models*, to more complex graph-structured *UML statecharts*, to syntax for meta-modelling languages like *Ecore*. Last but not least, EMFText has proven powerful enough to handle full-fledged programming languages: We used it in the *JaMoPP* project to create TS tooling for a Java 5 metamodel. Currently, there is ongoing work on other C-style languages like C#.

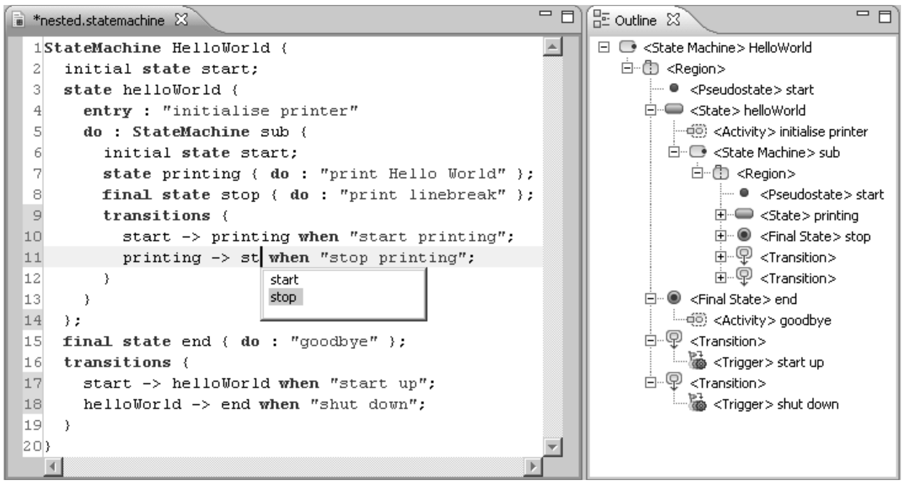
For the remaining sections in this paper, we focus on *UML statecharts* (Section 2) and *JaMoPP* (Section 3) as example applications of EMFText.

[1] see *Syntax-Zoo* on http://www.emftext.org/zoo

2 Textual Syntax for UML State Machines

Motivation In Model-driven Software Development (MDSD) many different modelling, languages are used and new (domain-specific) languages are defined constantly. The central point of a language definition is its metamodel. It is usually defined prior to the specification of a syntax. EMFText was developed to support exactly this scenario, where metamodels exist or are under development independent of the concrete language syntax. As an example for an existing metamodel, we look at UML state charts, which are one part of the large standardized UML Metamodel [4]. Note that this application exemplifies the definition of textual syntax for an existing metamodel with graphical syntax and (EMF-based) tooling.

Overview The CS specification for state charts is based on the metamodel from [4]. Therefore, a model defined in this syntax, as the one shown in Figure 1, can be processed in the same manner as a graphically defined state chart. When modelling with UML, EMFText can be used in collaboration with graphical UML editors and advantages of graphical and textual syntax can be combined. As one can see in the outline view on the right side of Figure 1, EMFText tightly integrates with EMF and reuses the icons for model elements that are also used in other UML editors. When clicking an element in the outline, the same element is highlighted in the text. This helps modellers familiar with UML to understand the textual syntax by exploring existing models (e.g., state charts printed into the text syntax by EMFText).



```
StateMachine HelloWorld {
  initial state start;
  state helloWorld {
    entry : "initialise printer"
    do : StateMachine sub {
      initial state start;
      state printing { do : "print Hello World" };
      final state stop { do : "print linebreak" };
      transitions {
        start -> printing when "start printing";
        printing -> st when "stop printing";
      }
    }
  };
  final state end { do : "goodbye" };
  transitions {
    start -> helloWorld when "start up";
    helloWorld -> end when "shut down";
  }
}
```

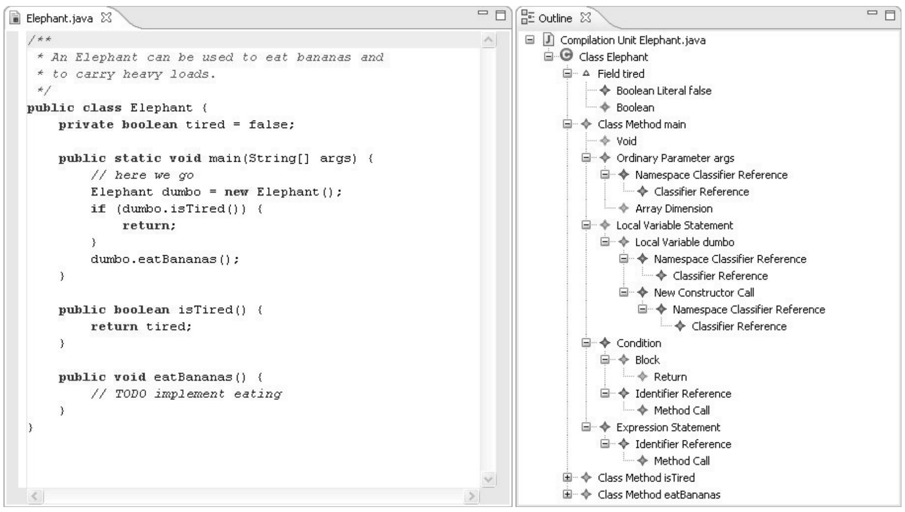
```
<State Machine> HelloWorld
  <Region>
    <Pseudostate> start
    <State> helloWorld
      <Activity> initialise printer
      <State Machine> sub
        <Region>
          <Pseudostate> start
          <State> printing
          <Final State> stop
          <Transition>
          <Transition>
    <Final State> end
      <Activity> goodbye
    <Transition>
      <Trigger> start up
    <Transition>
      <Trigger> shut down
```

Fig. 1. An UML state chart edited in the EMFText editor and the respective model tree

3 Models for Java Programs - JaMoPP

Motivation In MDSD models are refined and finally transformed into code using code generation techniques. While models are well-structured and formally defined in terms of metamodels, code generation tools (e.g., template engines) often create plain text that is eventually passed to a compiler to build the software defined by the input models. This loss of structure implies several drawbacks. First, there are no guarantees regarding syntactic or semantic correctness. Second, it is hard to trace which model elements triggered the creation of which parts of the code. If errors are found in the code, developers must figure out the affected parts of the model manually. To close this gap between models and code for the Java language, we developed Java Model Printer and Parser (JaMoPP) with the help of EMFText. Based on an Ecore metamodel, which defines the concepts of the Java language, and a CS specification of the Java syntax, JaMoPP treats Java programs like any other models in MDSD processes.

Overview JaMoPP consists of the aformentioned syntax definition from which a parser and a printer were generated by EMFText. Furthermore, to resolve cross-references between elements (e.g., targets of method calls or variable references), custom resolvers were implemented to reflect Java's scoping and referencing rules. For testing purposes, a huge set of test input files (e.g., the Eclipse source code) was passed to the JaMoPP parser. Figure 2 shows a simple Java class side by side with its model representation in the outline view.



```java
/**
 * An Elephant can be used to eat bananas and
 * to carry heavy loads.
 */
public class Elephant {
    private boolean tired = false;

    public static void main(String[] args) {
        // here we go
        Elephant dumbo = new Elephant();
        if (dumbo.isTired()) {
            return;
        }
        dumbo.eatBananas();
    }

    public boolean isTired() {
        return tired;
    }

    public void eatBananas() {
        // TODO implement eating
    }
}
```

```
Compilation Unit Elephant.java
  Class Elephant
    Field tired
      Boolean Literal false
      Boolean
    Class Method main
      Void
      Ordinary Parameter args
        Namespace Classifier Reference
          Classifier Reference
        Array Dimension
      Local Variable Statement
        Local Variable dumbo
          Namespace Classifier Reference
            Classifier Reference
          New Constructor Call
            Namespace Classifier Reference
              Classifier Reference
      Condition
        Block
          Return
        Identifier Reference
          Method Call
      Expression Statement
        Identifier Reference
          Method Call
    Class Method isTired
    Class Method eatBananas
```

Fig. 2. A Java class edited using the EMFText editor and the respective model tree

Acknowledgement

This research has been co-funded by the European Commission within the 6th Framework Programme project MODELPLEX #034081, by the German Research Foundation within the project HyperAdapt and by the German Ministry of Education and Research within the projects feasiPLe and SuReal.

References

1. TU Dresden: Software Technology Group: EMFText. http://emftext.org (2009)
2. Object Management Group: Human Usable Textual Notation (HUTN) Specification. Final Adopted Specification ptc/02-12-01 (2002)
3. Heidenreich, F., Johannes, J., Karol, S., Seifert, M., Wende, C.: Derivation and Refinement of Textual Syntax for Models. In: Proc. of ECMDA-FA, To appear (2009)
4. The Eclipse Foundation: EMF-based implementation of UML2 metamodel. http://www.eclipse.org/modeling/mdt/?project=uml2 (2008)

eXtreme Model-Driven Design with jABC

Demo track ¨Tools and Consultancy¨at ECMDA 2009

Christian Kubczak[1], Sven Jörges[1], Tiziana Margaria[2], and Bernhard Steffen[1]

[1] TU Dortmund, Chair of Programming Systems kubczak, jörges, steffen@cs.tu-dortmund.de
[2] Universität Potsdam, Chair of Software Engineering margaria@cs.uni-potsdam.de

Abstract. eXtreme Model-Driven Design (XMDD) is a new development paradigm designed to continuously involve the customer/application expert throughout the whole system's life cycle. In technical practice, user-level models are successively enriched and refined from the user perspective, until a sufficient level of detail is reached, at which elementary services can be implemented that solve tasks at the application level. The realization of the individual services should be typically simple: they are often based on functionality provided by third-party and by standard software systems. We demonstrate jABC, a flexible framework designed to support systematic development according to the XMDD paradigm. jABC allows (end-)users to develop service-oriented systems by composing reusable building blocks into flow graph structures. As a case study we present a heterogeneous service mashup that makes extensive use of today's service technologies like REST or Web services. Furthermore we introduce a temporal logic-based synthesis approach that automatically delivers process flows that conform to declarative formal specifications on the basis of semantic information annotated to the service components.

1 eXtreme Model-Driven Design with jABC

eXtreme Model-Driven Design (XMDD) is a new development paradigm designed to continuously involve the customer/application expert throughout the whole system's life cycle. In technical practice, user-level models are successively enriched and refined from the user perspective, until a sufficient level of detail is reached, where elementary services, solving tasks at the application level, can be implemented. The realization of the individual services should typically be simple and are often based on functionality provided by third-party and standard software systems. As the continuously enriched model is the central and sole artifact of this methodology, we also call this the ¨One-Thing Approach¨ [1].
jABC is a flexible framework designed to support systematic development according to the XMDD paradigm. It allows users to develop service-oriented systems by composing reusable building blocks into flow graph structures, called Service Logic Graphs (SLG). The building blocks are called Service Independent Building Blocks (SIBs), and may represent a single atomic service or also a whole subgraph (i.e. another SLG). Thus SLGs can be hierarchical, which facilitates the refinement along the lines of the One-Thing Approach, and which grants a high reusability not only of the building blocks,

but also of the models themselves, within larger systems. Besides SIBs and hierarchical SLGs, there are constraints which define the rules of a system and can be verified by formal methods like model checking. Finally, the modelled SLG can be compiled and/or synthesized to a running system (see Sect. **??** and Sect. 4.
Furthermore, an extensible set of jABC plugins provides additional functionality that adequately supports all the activities needed along the development lifecycle.
Typically, we distinguish different roles for the development process, depending on the various areas of expertise:

– Programming Experts, the software engineers responsible for the software infrastructure, the runtime environment for the compiled services, as well as the programming of the SIBs.
– Domain Modelling Experts: They build, use, modify and curate the underlying domain models, typically in some knowledge basis that can be expressed via ontologies. These experts classify the SIBs, typically according to technical criteria like their version or specific hardware or software requirements, their origin (where they were developed) and according to their intent for a given application area.
– Application Experts: They develop concrete applications just by defining their Service Logic structure. This happens without programming: they graphically combine building blocks into coarse-granular flow graphs (the SLGs) and graphically configure the data path.

End Users may customize a given (global) service according to their needs by parametrization and specialization.

Fig. 1 shows a screenshot of jABC's user interface, which can be personalized to serve all roles named above, except of the programming experts. The GUI consists of three main parts (indicated by the numbers):

1. the project & SIB explorers, which enable the user to browse available jABC projects and the library of building blocks that can be used for modelling,
2. the graph canvas, which is used for composing SLGs, and
3. the inspectors, which provide detailed information about selected SIBs and may be used by plugins to add further functionality.

2 Full Application Code Generation with Genesys

Once the SLG of a jABC application is fully designed and the SIBs are all implemented, it is ready for deployment. The SLG is transformed into an executable and deployable piece of code in a desired programming language resp. for a desired target platform.
This is supported by the jABC code generation library Genesys [2], which is available via a jABC plugin. The library's philosophy is not to be an ¨all-in-one device suitable for every purpose¨, but rather to provide readymade, highly specialized and domain-specific code generators for different target formats like Java, C#, BPEL, Web Services etc.
If a custom code generator is required for a specific domain, Genesys also offers a powerful methodology for easily and quickly creating a new code generator. The construction and evolution of the code generation library is itself based on the application

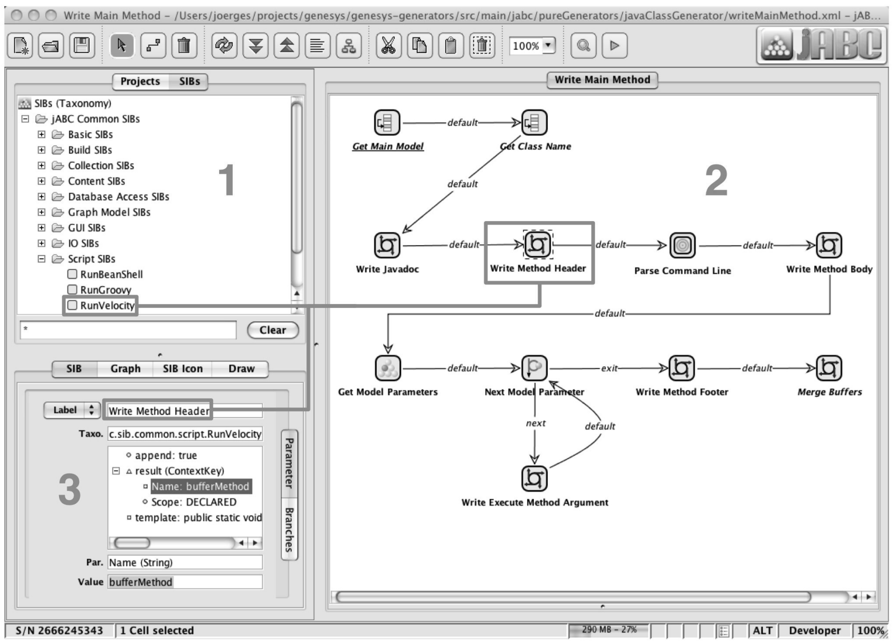


Fig. 1. The jABC User Interface

of the XMDD paradigm, i.e. that the code generator are not programmed by hand, but all modelled within jABC. This way developers of new generators immediately benefit from already implemented generators by reusing SIBs as well as whole models. As currently most of the generators in the library are template-based, the easiest way to build a new generator often is to simply modify the templates of an existing generator accordingly. Furthermore, the generators can be easily verified using the jABC verification mechanisms, thus assuring a certain quality of the generation process.

In contrast to existing MDSD/MDA code generation frameworks like AndroMDA or openArchitectureWare, Genesys code generators always produce complete source code, which does not require any manual editing by a developer. For instance, AndroMDA mostly produces stubs or skeletons which have to be manually completed in order to be executable.

This fact is especially due to the limitations of the usage of UML as the modelling notation for most MDSD/MDA frameworks. UML is well suitable to describe technical, static and infrastructural aspects of a system interesting for programming experts, but has clear weaknesses when it comes to the dynamic parts, where jABC and Genesys play to their strengths.

Thus existing MDSD/MDA solutions and jABC/Genesys can perfectly be used in conjunction to capture the static as well as the dynamic aspects of a system, and to generate full application source code without the need of manual post-editing.

3 Heterogeneous Service Mashups

When talking about mashups especially in terms of service oriented computing one usually means the combination of diverse tools from different providers to end up with a completely new and more powerful application making extensive use of single features from each reused service. With its intuitive but powerful user interface and modeling concepts jABC is to combine different technologies of appropriate vendors strictly following the concepts of Service Oriented Architecture (SOA) [3,4,5] and User-Centric Modeling [1,6].
During demonstration we show how jABC can be used to easily create mashups using local service components, web services and REST. As a showcase we pick some well known providers like Google, Amazon, Flickr & Wikipedia.
When talking about mashups especially in terms of service oriented computing one usually means the combination of diverse tools from different providers to end up with a completely new and more powerful application making extensive use of single features from each reused service. Big internet companies and vendors thereby support programmers by providing web services like through the Amazon Web Service Developer Connection [1] or APIs as the Google Web Tool Kit [2]. These technologies are easy to use for programmers on the code level but inappropriate for experts or end users trying to combine their own special service mashup. To overcome this we use jABC [7,8], a service oriented framework for graphical orchestration and choreography of tools and services.

3.1 Service Handling.

As stated in Sect. 1 services within the jABC framework are represented as SIBs which are grouped together by taxonomic descriptions inside a service library. Each SIB has a collection of parameters to take input and generate output data (see bottom left corner of Fig. 2) as well as a set of outgoing branch labels to define the control flow within an SLG. Typically the implementation of a single SIB is split up into two parts forming the final service component: A SIB container and a SIB adapter. The whole service functionality is thereby encapsulated inside the latter. As shown in Fig. 3 the SIB itself just invokes the appropriate service routine and denominates a corresponding adapter which can of course serve more than only one SIB. Using this technique it is therefore possible to group service functionality together into one powerful adapter serving a whole set of different SIBs. This concept is completely hidden to the end user. Speaking technically each SIB and each adapter always correspond to an underlying Java class implementing one or more interfaces depending on the plugin support a component would like to offer. For example, to make a SIB an executable component one has to implement a code fragment telling the service what to do when invoked by the runtime environment. Reusability is one of the main concepts of Service Oriented Architecture jABC is based upon. Dealing with SIBs like this, the library grows over time leading to an increasing flexibility of component usage and service mashups. In Sect. 3.2 we

[1] http://aws.amazon.com
[2] http://code.google.com/webtoolkit

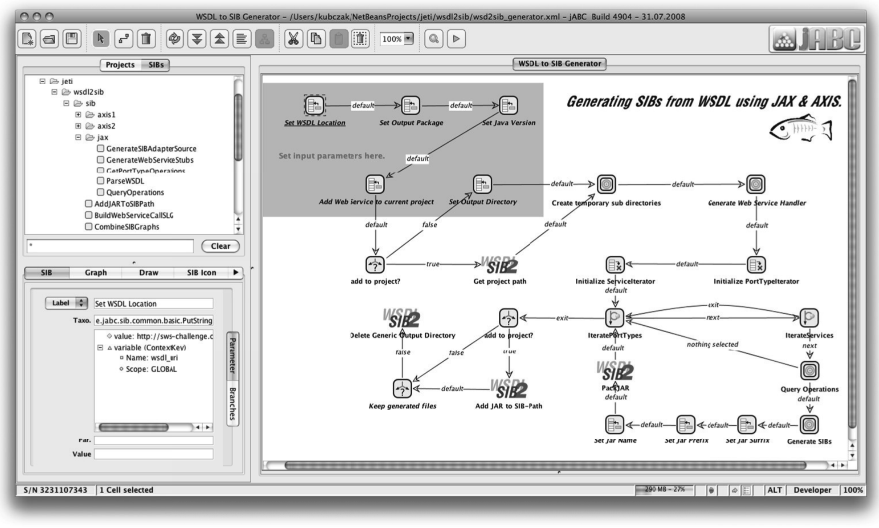


Fig. 2. jETI's web service import process.

show how to import service components with less and partly without programming knowledge.

3.2 Remote Service Technology.

As briefly described above jABC offers the possibility of application modeling. For the integration of remote services and tools we introduced jETI [9] as an enhancement framework to jABC. jETI provides a powerful but easy to use framework to simplify both, the integration and usage of remote tools and services. Fig. 4 gives an overview of jETI's architecture. Tools and services are provided by vendors using a toolserver or *Service Provider Site* (SPS) depicted in the top right corner. The components are hosted on the server and described using an XML formated document. Service provider specify what command should be executed on the server and which parameters it has to take to operate successfully. Afterwards the SPS can be registered at a *Component Server* (CS) managing a set of toolservers. Thereby a CS handles certificates, policies and general information about service provision (like locations and redundant services) and acts as the main interface to the client which in fact is a plugin to jABC. During application modeling the user is able to browse services hosted on various SPS by using this client and import SIBs representing the provided tools. These SIBs are thereby generated by the CS using the tool descriptions stored at the different SPS. Nevertheless sometimes a service has to be even more open or powerful to be integrated in complex process flows or service environments. During the last few years REST and web services have become standard technologies for this purpose. Today most vendors inside the mashup community provide their functionality as web services to the users.

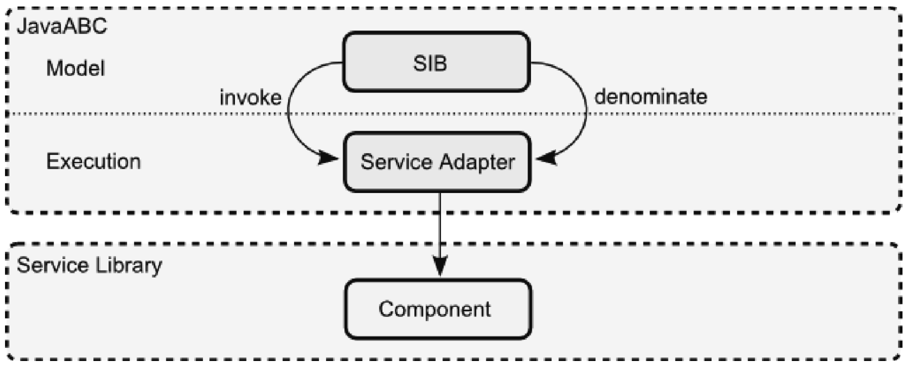


Fig. 3. SIB adapter concept.

When talking about web services one usually means an application defining its interface by a Web Service Description Language (WSDL) document [10] and handling communication via SOAP [11]. To handle web services inside jABC the plugin of jETI offers a set of features. As WSDL is based on plain XML the description is a formal specification of this service which can be used to automatically generate client site code to invoke the appropriate functions of an application. To this purpose there exists a set of toolkits and frameworks. Dealing with Java which is the underlying technology of jABC and jETI the most common of them are AXIS, AXIS2 and JAX-WS [3]. Those frameworks provide mechanisms to automatically generate client stubs and data type bindings out of a given WSDL specification. In the following subsections jETI's capability to import and export web services based on these three frameworks are described in detail.

Web Service Import. The basic idea of web service usage inside jABC is to have a single SIB for each operation a web service provides. As we intend to provide a convenient solution jETI offers an automatic import for web services using three different data binding frameworks underneath (AXIS, AXIS2, JAX-WS). Regarding the concepts of SOA this functionality is not coded from the scratch using Java or any other programming language but developed under service oriented aspects using the modeling framework itself. Modeling the workflow makes the application much clearer to the developer during implementation and reusing services helps to save time during development. The final process is described as follows. Fig. 2 shows the web service import process modeled inside jABC.

Set WSDL Location, Set Output Package, Set Java Version, Add Web Service to current project and *Set Output Directory.* First some input parameters have to be set which is done using the first five SIBs inside the grey area in the top left corner. The latter is only needed if the generated SIBs are not imported directly to the current working project.

[3]http://ws.apache.org/axis, http://ws.apache.org/axis2, https://jax-ws.dev.java.net

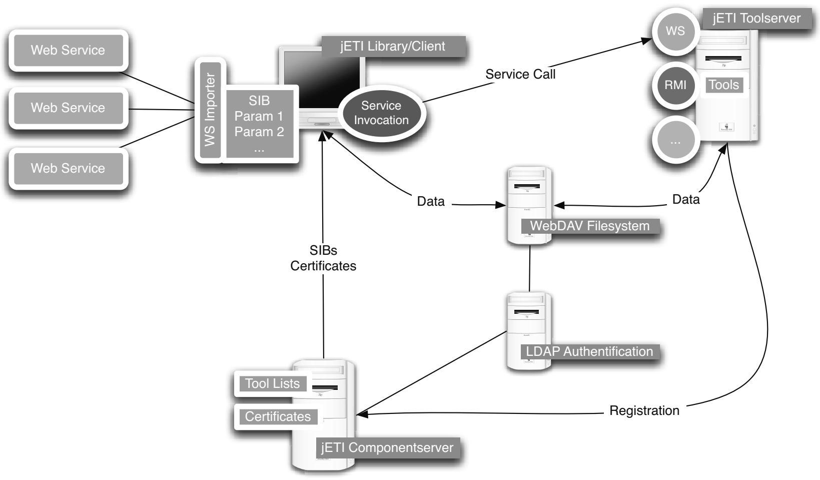


Fig. 4. Architecture of jETI.

Afterwards some temporary directories are created using a trivial subgraph hidden by the hierarchical so called Graph SIB *Create temporary sub directories.* As this sub-structure is just a linear chain of services creating directories on the hard disc it is not depicted here.

The next Graph SIB (*Generate Web Service Handler*) encapsulates the generation of web service handler classes using either AXIS, AXIS2 or JAX-WS. The subgraph is shown in Fig. 5.

First there is a decision inside *Use cascade mode* whether to use a specific framework (this is using the path on the right including *Set Generation Framework* and *Switch-Framework*) or to try a cascading style for choosing a framework. The latter starts trying the JAX-WS environment and goes down with AXIS2 and AXIS if an error occurs during generation. This could be because of incompatible features a WSDL defines and cannot be handled equally by every framework for instance.

The generated handler source files are compiled during runtime by *Compile* using the Java version defined in the parent graph structure at the beginning of the import process (see Fig. 2, top left corner).

AXIS and AXIS2 also provide a list of all operations defined inside the WSDL specification while generating the handler code. As JAX-WS does not, in case of its usage there has to be a separate parsing of the WSDL done by the *ParseWSDL* service.

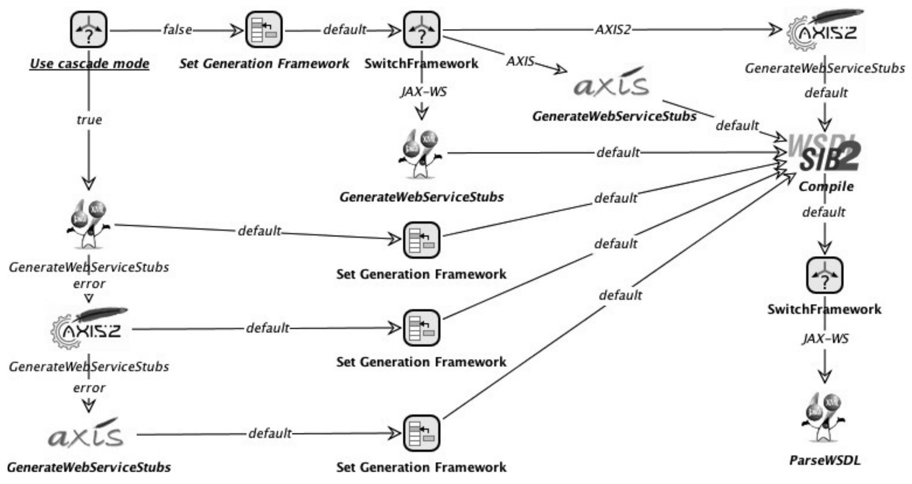


Fig. 5. The subgraph to generate handler stubs.

After retrieving all the operations the subgraph returns to the parent process initializing to loop iterators by invoking *Initialize PortTypeIterator* and *Initialize ServiceIterator.* The following loop now iterates over all the defined Services and PortTypes inside a WSDL (usually one of each kind) and pops up a GUI querying the user for input to specify which operations of a web service to import. This is done by a trivial subgraph inside *Query Operations* which is not depicted due to the fact that it just defines a minimal workflow to show a window containing a list.

During the next step *Generate SIBs* is invoked. The underlying substructure is shown in Fig. 6.

First the output packages to generate the SIBs in are defined.

Afterwards the adapter (see Sect. 3.1) classes are generated and compiled containing all the operations chosen for import before in the parent workflow.

Following the adapter generation the SIB sources are generated inside *GenerateSIB-Source* by using API calls to jABC's SIBCreator plugin.

As each of the used WS frameworks generates single Java objects for a service, a port type and the operation itself and the SIBCreator provides a single SIB for each of these classes the next step is to build a linear SLG out of the generated SIBs calling the service which contains the port type used to invoke the operation. This is done by invoking *BuildWebServiceCallSIBGraph* .

Afterwards information about input and output type bindings of the imported web service operation is collected and written as annotations inside the generated SIB sources (*StartVelocityEngine* and *GenerateTypeInformationBaseClass* .

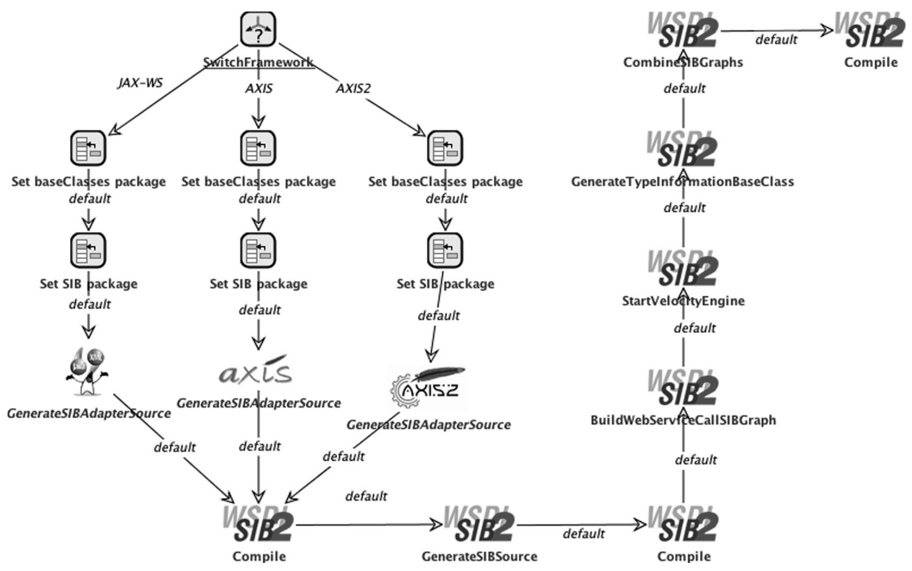


Fig. 6. The subgraph to generate SIBs out of a given WSDL.

The subprocess ends by combing the generated web service call graph into only one single SIB (*CombineSIBGraphs*) which is then compiled using the *Compile* service once again.

Returning to the parent workflow some output parameters are set (*Set Jar Suffix*, *Set Jar Prefix* and *Set Jar Name*) and the generated and compile classes are packed inside a JAR archive invoking *PackJAR*.

If there are no more port types (which is the usual case as stated before) the gained JAR is optionally added to the current working project's classpath (*Add JAR to SIB-Path*) and the generated source files are removed if one not wants to explicitly keep them (*Keep generated file*, *Delete Generic Output Directory*).

For a convenient usage of the import mechanism inside jABC, the Genesys plugin [2,12] is used to generate Java code out of the above workflow which is then triggered through jABC's GUI. Therefore a user basically has to provide a WSDL location like a URL or a file and automatically gets a set of SIBs representing the operations of the appropriate web service.

Web Service Export. Analogous to the web service import described in Sect. 3.2 jETI is able to handle web service exports meaning to generate a running web service out of a modeled process. In general there are two approaches for implementing web services named *code first* and *contract first*:

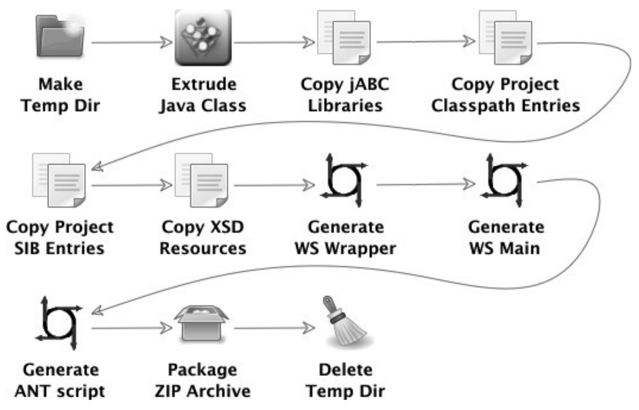


Fig. 7. jETI's web service generation process.

- In a code first approach, some implemented functionality is available that should be published as a web service. The related WSDL file is ideally generated by the used web services toolkit or framework like AXIS, AXIS2 or JAX-WS at deployment.
- In a contract first approach, one starts by writing a WSDL by hand or tool-supported and then generates the web services stubs and skeletons with a generation tool usually provided by the used framework.

To start using contract first and writing a WSDL document one needs to be at least familiar with programming concepts and XML syntax while one ends up with source file stubs to be filled with functional programming code. As jABC is intended to be used by non-technical users and one usually starts with modeling an application, jETI only support code first which is accessible as a plugin using jABC's GUI like the web service import. The underlying export functionality is again modeled using the jABC itself. Fig. 7 shows the export process as an SLG.

First, *Make Temp Dir* creates a dedicated temporary directory for the work of the subsequent SIBs. *This is a trivial subgraph like the one used during the above import SLG.*

Afterwards the Genesys plugin is used to extrude Java Code out of the modeled application invoking *Extrude Java Class*. The result is a Java class that performs the same execution of the model as it would be done by the jABC.

As the generated Java class depends on some core jABC libraries *Copy jABC Libraries* is used to put them together for the resulting web service.

Copy Project Classpath Entries does the same with user specified libraries needed for the execution of the modeled workflow.

Finally *Copy Project SIB Entries* copies all the SIBs used inside the SLG representing the future web service.

If the web services interface definition contains more than simple XSD [13] types, one can provide separate XSD type denition documents which is done by the *Copy XSD Resources* SIB.

The next steps consist of generating the web service wrapper and main classes (*Generate WS Wrapper* and *Generate WS Main*).

The process is afterwards finished by providing a corresponding Apache ANT [4] script file (*Generate ANT script*) containing all the necessary information to start the resulting web service and by packing the generated files inside a ZIP archive for convenient provision (*Package ZIP Archive*).

To clean up the temporary directory *Delete Temp Dir* is invoked as the final SIB.

The web service generation process results in a complete web service, with all its sources and dependencies being generated and packed together. As the underlying framework JAX-WS is used as it not only provides a complete set of web service stubs, handlers and a WSDL interface description (like AXIS or AXIS2 would of course) but also contains a minimal but fully working HTTP server to host the resulting service. Unlike AXIS/AXIS2 where the user has to deploy the final web service onto a dedicated web server, one can easily startup the service by running a command from the ANT script generated during the export of the service's SLG.

REST Services. In addition to web services defining their interface by using a WSDL document a common technology is the concept of Representational State Transfer (REST) [14]. Web sites, pictures or servlet code can be seen as resources which are accessible only using a simple URL as direct access is forbidden. Now, if a client, which can be a browser for instance, navigates to the location, a representation of the underlying resource is sent back as an HTML document for example. If this hypertext document contains links to other document the client could change its status by navigating to one of them. Therefore using the representation of a resource a transfer of the client's state is performed. As REST uses a generic interface based on semantics of the HTTP protocol there is no formal specification like WSDL for those services. Operations to be performed are restricted to request the representation of a resource (*GET*), to create a new resource (*PUT*), to add content to a resource like variables (*POST*) and to remove a resource (*DELETE*). Unlike WSDL specified web services REST services in general provide no formal definitions of their used variables and data format. Integration of such tools as SIBs within a process therefore assume the user to have a basic knowledge of their functionality which is usually provided through natural language documentations. The *Google Maps Geocode* service [5] takes a natural address and provides the relating longitude and latitude as a returning string for example whereas the *Geonames* service [6] takes a latitude and longitude and returns an XML document containing a set of nearby

[4]http://ant.apache.org
[5]http://maps.google.com
[6]http://www.geonames.org

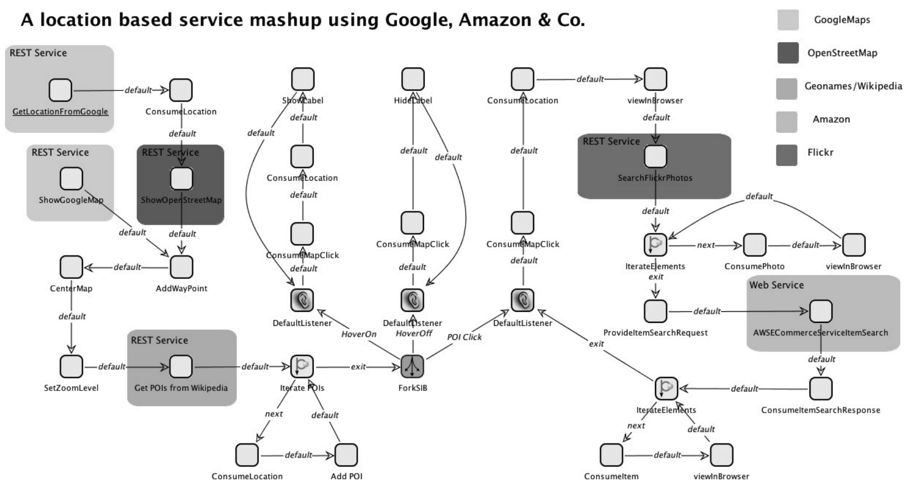


Fig. 8. Mashup with Google, OpenStreetMaps, Geonames, Wikipedia, Amazon, Flickr.

points of interest with its details. As these information significantly differ from one service to another integration of REST services inside jABC is done by implementing SIBs from the scratch invoking the appropriate method with a specific set of variables and returning and manipulating the data returned by the service.

3.3 Google, Amazon & Co Service Mashup.

This case study mashes up some of todays most used web services on the internet. As an overview we divide the whole process shown in Fig. 8 into 3 parts:

1. We locate and display an address in *Google Maps* or *OpenStreetMap* [7] (services marked green and yellow).
2. We try to find nearby points of interest (POI) using *GeoNames* (pink service).
3. Afterwards when clicked on an POI we look for a relating *WikiPedia* [8] article, Picture on *Flickr* [9] and search for likely interesting products on *Amazon* [10] (red and blue services).

First a REST service from Google Maps is called to get a geocode location (which is longitude and latitude) to a given physical address. As the SIB returns a complex type *Location* the data encapsulated by this type has to be extracted using the *ConsumeLocation* SIB.

[7]http://openstreetmap.org
[8]http://www.wikipedia.org
[9]http://flickr.com
[10]http://www.amazon.com

Afterwards an OpenStreetMap is displayed inside a window by invoking the REST service SIB *ShowOpenStreetMap*.

A waypoint is added to the map at the position from the beginning and the map view is centered around this point. The next step defines a zoom level for the displayed part of the map.

While we now see a detailed view of the specified address a REST Service provided by GeoNames is called returning a given number of interesting waypoints added to the map within the following iteration. The returned Location types of the POIs also contain information about relating Wikipedia entries.

After this a special control flow SIB called *ForkSIB* is invoked. It splits the current execution into several new threads running parallel at the same time. All these new execution paths lead to a *DefaultListener* SIB each waiting for a specified event to occur inside the map.

The most left wait for a mouse hover on a waypoint inside the map and displays the name of this waypoint by calling *ShowLabel*. The second thread wait for a mouse hover off and hides the previously showed title by invoking *HideLabel*. The most right execution path consumes a mouse click on any waypoint reading the data out of the returned Location type (see above) and opens up a web browser to display the relating Wikipedia entry.

By executing *SearchFlickrPhotos* during the next step, a given number of photos tagged with the title name of the waypoint is returned and diplayed inside the following loop using *viewInBrowser* once again.

The last service to call is *AWSECommerceServiceItemSearch* which is an automatically imported web service provided by Amazon returning a given number of items on sale relating to a specified searchRequest. As the latter is also encapsulated inside a complex type (see *Location* above) a SIB providing it has to be called first *ProvideSearchRequest*. Same is for the returned item information read out by the *ConsumeItemSearchResponse* SIB. All found items are displayed again by invoking *viewInBrowser* inside an execution loop. Finally the *DefaultListener* SIB waits for the next mouse click until the application is quit by the user or the appropriate event occurs.

After modeling the application as stated above one is able to generate different kinds of source code or a web service out of it. One could easily extrude machine code for a mobile handheld for example using jABC's Genesys code generator plugin and making extensive use of GPS and other features of a specific device.

3.4 Message.

Using jABC/jETI to implement a mashup like this case study has several advantages than dealing with todays common technologies like scripting or real programming languages even if simplified by toolkits and clean APIs.First of all even non programmers

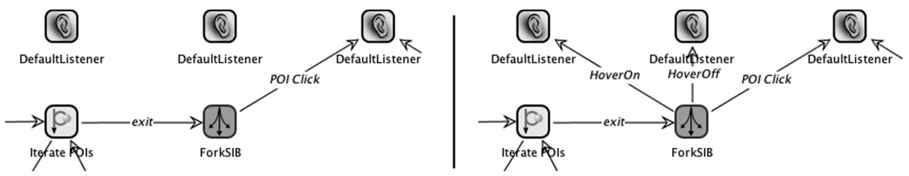


Fig. 9. An extract of the mouse hover reaction.

can access the technology by graphically designing their appropriate workflows.Second, once a (web) service was implemented as a SIB or imported into jABC one gains a high degree of reusability by sharing the SIB library. Any other mashup using single features of the presented application can fall back on the appropriate SIB without changing a single line of code. Third, convertibility of the modeled application is very intuitive and easy to realize. On the left hand side of Fig. /reffig:hover the three listener SIBs to react to mouse hover on/off and POI clicks are shown. If one wants to deactivate the mouse hover effect for instance it is realized just by removing the appropriate branches like depicted on the right hand side of Fig. 9. Same could be done for changing the flow of an application by simply make a branch point to another service, for example changing the mashup to show information about a POI while hovering over a waypoint instead of clicking it.
A more detailed view on service mashups with jABC and the case study shown above is currently under review and will soon be published at [15].

4 Synthesizing process flows (Model generation)

jABC features a fully integrated synthesis framework to automatically generate process flows by specifying temporal logic constraints. As to show business related service composition in context of synthesis we will demonstrate our contribution to mediation scenarios of the Semantic Web Service Challenge (SWSC) [16] (see workshops of ECMDA'09) whereas participants have to implement a middle/communication layer that mediates protocols and data between two business parties.
We show how to automatically generate the SWSC mediators service logic, as a declarative alternative to a typical hands-on approach like Sect. 3. Here we use an LTL model generation algorithm which works on the basis of the existing jABC library of available services (SIB library) already introduced in Sect. 1. It uses an enhanced description of their semantics that is given in terms of a taxonomic classication of their behaviour (modules) and abstract interfaces/messages (types). The resulting approach is a forward synthesis algorithm that users can congure to provide the set of shortest, or cycle-free, or all orchestrations, that satisfy the given LTL specication.
The synthesis tool takes a text le containing the knowledge base as an input: the module and type taxonomy, the module descriptions, and some documentation for the integrated hypertext system. It is steered from the jABC GUI (see Fig. 1). There, users can input the SLTL formulas that describe the goal and can ask for different kinds of

activity name	input type	output type	description
`Mediator`			Maps RosettaNet messages to the backend
`startService`	$\{true\}$	*PurOrderReq*	Receives a purchase order request message
`obtCustomerID`	*PurOrderReq*	*SearchString*	Obtains a customer search string from the req. message
`createOrderUCID`	*CustomerObject*	*CustomerID*	Gets the customer id out of the customer object
`buildTuple`	*OrderID*	*Tuple*	Builds a tuple out of the id of the orderID and the POR
`sendLineItem`	*Tuple*	*LineItem*	Gets a LineItem incl. orderID, articleID and quantity
`closeOrderMed`	*SubmConfObj*	*OrderID*	Closes an order on the mediator side
`confirmLIOperation`	*OrderConfObj*	*PurOrderCon*	Receives a conf. or ref. of a LineItem and sends a conf.
`Moon`			The backend system
`searchCustomer`	*SearchString*	*CustomerObject*	Gets a customer object out of the backend database
`createOrder`	*CustomerID*	*OrderID*	Creates an order
`addLineItem`	*LineItem*	*SubmConfObj*	Submits a line item to the backend database
`closeOrderMoon`	*OrderID*	*TimeoutOut*	Closes an order on the backend side
`confRefLineItem`	*Timeout*	*orderConfObj*	Sends a conf. or ref. of a prev. subm. LineItem

Table 1. The SWS mediation Activities

activity name	input type	output type	description
`Mediator`			Maps RosettaNet messages to the backend
`buildTuple`	*Order*	*Tuple*	Builds a tuple out of the id of the orderID and the POR
`closeOrderMed`	*SubmConfObj*	*Order*	Closes an order on the mediator side
`confirmLIOperation`	*Order*	*PurOrderCon*	Receives a conf. or ref. of a LineItem and sends a conf.
`Moon`			The backend system
`createOrder`	*CustomerID*	*Order*	Creates an order
`closeOrderMoon`	*Order*	*TimeoutOut*	Closes an order on the backend side
`confRefLineItem`	*Timeout*	*Order*	Sends a conf. or ref. of a prev. subm. LineItem

Table 2. The SWS mediation Activities with abstract `Order`

solutions. The tool produces a graphical visualization of the satisfying plans (module compositions), which can be executed, if the corresponding module implementations are already available, or they can be exported for later use. The knowledge basis implicitly describes the set of all legal executions. We call it conguration universe, and it contains all the compatible module compositions with respect to the given taxonomies and to the given collection of modules.
The typical user interaction foresees a successive renement of the declarative specication by starting with an initial, intuitive specication, that asks typically for shortest or minimal solutions, and using the graphical output for inspection and renement.
The ongoing Sematic Web Service Challenge proposes a number of increasingly complex scenarios for workflow mediation and service discovery. We use here the technology presented in [17] [11] to synthesise a workflow that realizes the communication layer for the Challenge's initial mediation scenario.

[11]The referenced publication describing the underlaying technology is provided in addition to the SWS community.

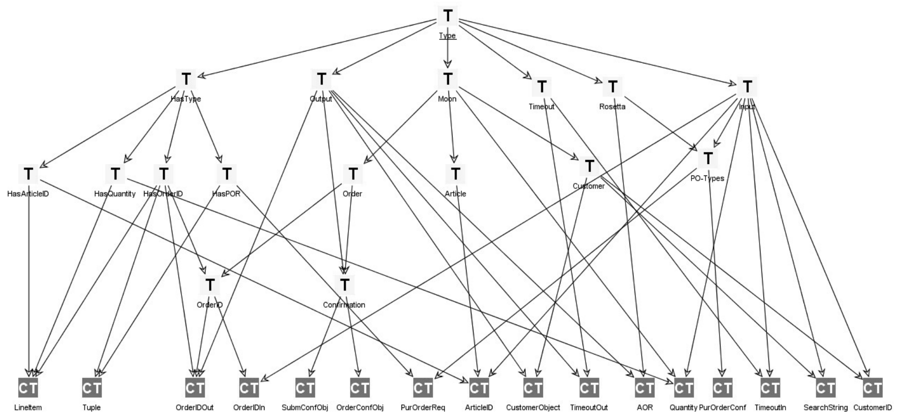


Fig. 10. The SWS Challenge Mediator Type Taxonomy

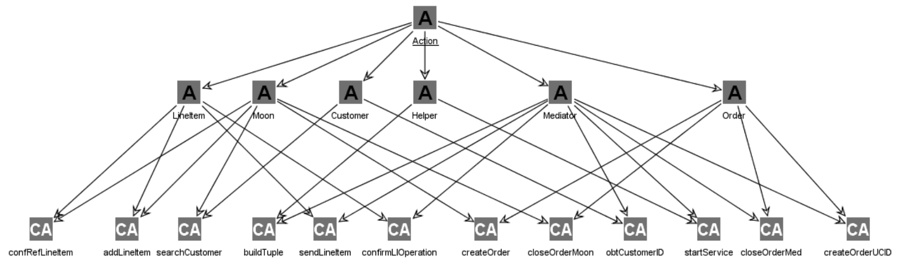


Fig. 11. The SWS Challenge Mediator Action Taxonomy

4.1 The concrete mediator workflow.

Of course we can easily define the concrete process within our jABC modelling framework, as we have shown in the past [18]. To provide a more flexible solution frameworks, especially to accommodate later specification changes on the backend side or the data flow, we synthesize the whole mediator using the jETI synthesis technology introduced in [17]. We proceed exactly along the lines already presented in that paper. As in the example described there, Tab. 1 shows the activities identified within the system.

The taxonomies regarding the mediation scenario are shown in Fig. 10 (Type Taxonomies) and Fig. 11 (Action Taxonomies).

We ask for a workflow that satisfies the following requirement:

Use the mediator service to produce a Purchase Order Confirmation.

The corresponding formal specification is simple: we need to start the service (activity startService) and reach the result `PurOrderCon` (a type). We may simply write: `(startService < PurOrderCon)` The jABC process model shown in Fig. 12

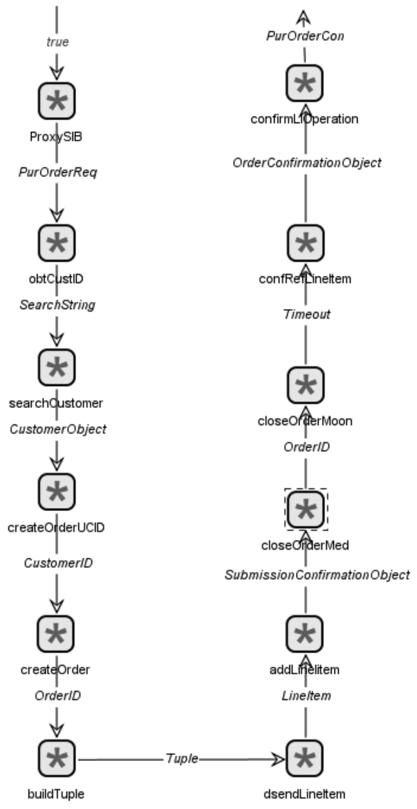


Fig. 12. The synthesised SWS mediator (standard)

resembles very closely the expected required solution.
This is in fact the only solution if we adopt the very fine granular model of the types shown in Table 1, which is a natural choice given the SWS Challenge problem description (Here we use the abstract type names to model de facto almost the SOS-level granularity: we distinguish for instance an `OrderID` from an `OrderConfObject`, modelling the described application domain at the concrete level of single datatypes and objects - a direct rendering of what happens in the memory and in the heap. This is however already a technical view.

4.2 Using Abstraction and Constraints.

For a specifier and definer of the business domain it is much more realistic to say that the activities concerned with orders work on an `Order` type, which is the business-level abstraction for these objects and records, and leave the distinctions to a problem-specific specification of the desired solutions via constraints.

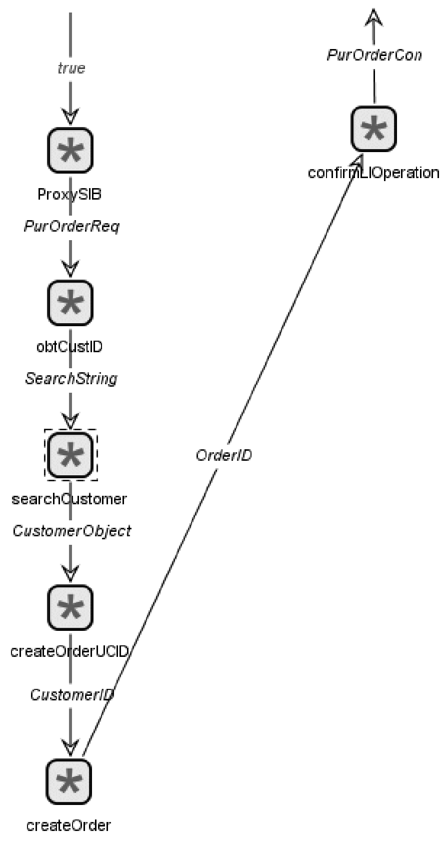


Fig. 13. Using Loose Types: the new solution

The taxonomy design and module specification decides the balance between concreteness and flexibility. In this case we change the definition of the modules that deal with orders as show in Tab. 2. We can be as concrete as we wish, or as abstract and generic as we wish, and have a semantics or application domain modelling-driven description that determines how much flexibility we build in into our solutions. At the one extreme we have very specific types, as fine granular as an SOS description. Then solutions are type-determined, and basically render the concrete labelled transition system underlying the manually programmed solution. At the other extreme we exploit looseness in the taxonomies and in the module descriptions, and the solutions are constraint-determined, in order to reach the same precision.
No matter the choice, the algorithm covers the whole spectrum, leaving it free to the application domain designer to determine where to be precise and where to be loose, leaving space for exploring alternatives.

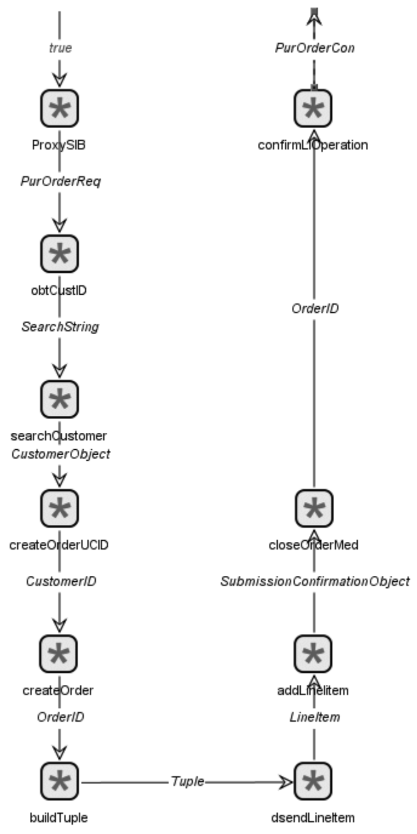


Fig. 14. Adding a LineItem: the new solution

Loose Solution If we now solve the planning problem with the modified module description and the original goal, we obtain a much shorter solution, shown in Fig. 13. This is due to the fact that these module specifications now refer to the abstract type *Order*. As a consequence, closeOrderMoon is a suitable direct successor of createOrder. This solution corresponds to the degenerate workflow where an empty order is sent. Since in the normal case orders contain items, we need to be more precise in the specification of the solution, adding constraints. If we just know that the items are referred to via the LineItem type, we may simply refine our goal as follows:

```
(startService < LineItem < PurOrderCon)
```

This way, we have added as additional intermediate goal the use of a LineItem type. Accordingly, at least one of the activities {addLineItem, sendlineItem} must appear in the required minimal workflow. We see the result in Fig. 14: the solution coincides with the previous one till the createOrder activity, then the type mediator buildTuple is added, after which sendLineItem satisfies the intermediate goal. The remaining

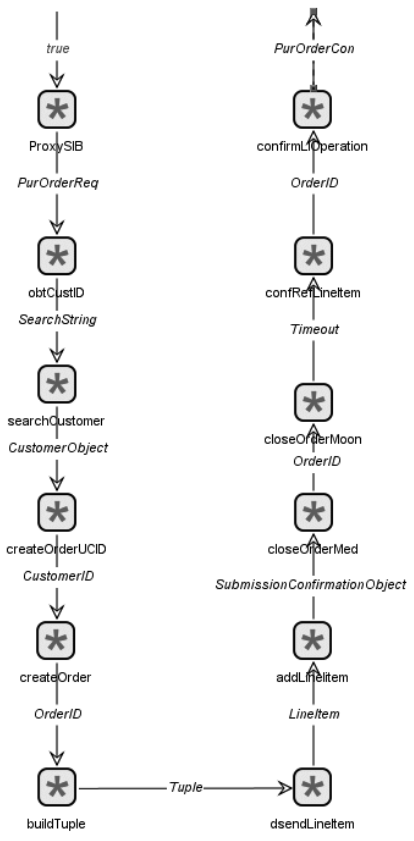


Fig. 15. Adding a Confirmation: the complete loose solution

constraint at that point is simply the reaching of the final type PurOrderCon, which is done by generating the sequence CloseOrderMediator followed by CloseOrder. This solution however corresponds only to the first webservice realizing the mediator. There is a subsequent second service that realizes the confirmation part of the mediator.
To obtain this part as well, we have to specify that we need to see a confirmation, e.g. as confRefLineItem activity:

```
(startService < LineItem <
      confRefLineItem <PurOrderCon)
```

This generates the solution of Fig. 15, which includes also the rest of the sequence shown in Fig. 12.

Configuration Universe In this quite constrained example it is not difficult also to generate all the possible solutions: the space of all the solution (the configuration universe)

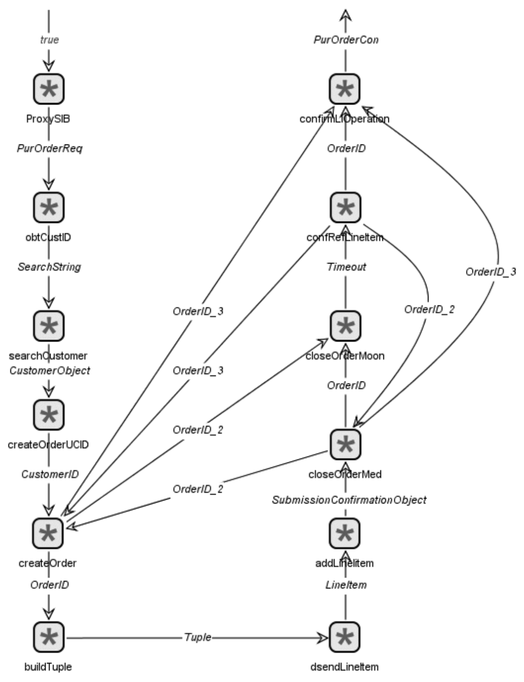


Fig. 16. The Configuration Universe

is shown in Fig. 16.

The synthesis and the SWS case study example presented above is decribed in more detail in [19].

References

1. Cardoso, J., van der Aalst, W., eds.: Business Process Modelling in the jABC: The One-Thing Approach. IGI Global (2009)
2. Jörges, S., Margaria, T., Steffen, B.: Genesys: Service-oriented construction of property conform code generators. Innovations in System and Software Engineering - a NASA Journal (October 2008)
3. Margolis, B.: SOA for the Business Developer: Concepts, BPEL, and SCA. Mc Press (May 2007)
4. Margaria, T., Steffen, B., Reitenspieß, M.: Service-oriented design: The roots. In: ICSOC. (2005) 450–464
5. Jung, G., Margaria, T., Nagel, R., Schubert, W., Steffen, B., Voigt, H.: SCA and jABC: Bringing a service-oriented paradigm to web-service construction. In: ISoLA'08, Proc. 3rd Int. Symp. on Leveraging Applications of Formal Methods, Verification, and Validation (CCIS). Volume 17., Springer Verlag (October 2008)

6. Margaria, T., Steffen, B.: Agile it: Thinking in user-centric models. In: ISoLA'08, Proc. 3rd Int. Symp. on Leveraging Applications of Formal Methods, Verification, and Validation (CCIS). Volume 17., Springer Verlag (October 2008)
7. Jörges, S., Kubczak, C., Nagel, R., Margaria, T., Steffen, B.: Model-driven development with the jABC. In: HVC - IBM Haifa Verification Conference. LNCS, Haifa, Israel, IBM, Springer Verlag (October 23-26 2006)
8. Margaria, T., Steffen, B.: Service engineering: Linking business and it. IEEE Computer, issue for the 60th anniversary of the Computer Society (October 2006) 53–63
9. Margaria, T., Nagel, R., Steffen, B.: jETI: A tool for remote tool integration. In: TACAS. (2005) 557–562
10. : Web Service Description Language. (2008) `http://www.w3.org/TR/wsdl`.
11. : SOAP. (2008) `http://www.w3.org/TR/soap/`.
12. Jörges, S., Kubczak, C., Pageau, F., Margaria, T.: Model driven design of reliable robot control programs using the jabc. In: Proc. EASe '07. (March 2007) 137–148
13. : XML Schemas. (2008) `http://www.w3.org/XML/Schema`.
14. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures - Doctoral dissertation. University of California (2000) `http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`.
15. Kubczak, C., Doedt, M., Margaria, T., Steffen, B.: From programming to planning: An automatic service mashup approach using jACB/jETI. InTech.org (2009, to be released) `http://intechweb.org`.
16. : Semantic Web Service Challenge. (2009) `http://www.sws-challenge.org`.
17. Margaria, T., Steffen, B.: LTL guided planning: Revisiting automatic tool composition in ETI. In: SEW: 31st Annual Software Engineering Workshop, IEEE Computer Society Press (6-8 March 2007)
18. Kubczak, C., Margaria, T., Steffen, B., Nagel, R.: Service-oriented Mediation with jABC/jETI. Springer Verlag (2008)
19. Margaria, T., Bakera, M., Kubczak, C., Naujokat, S., Steffen, B.: Automatic Generation of the SWS-Challenge Mediator with jABC/ABC. Springer Verlag (2008)

FURCAS: View Based Textual Modelling

Thomas Goldschmidt[1], Steffen Becker[1], Axel Uhl[2]

[1] FZI Research Center for Information Technology
Karlsruhe, Germany
{goldschmidt, sbecker}@fzi.de
[2] SAP AG
Walldorf, Germany
axel.uhl@sap.com

Abstract. Textual concrete syntaxes for models are beneficial for many reasons. They foster usability and productivity because of their fast editing style, their usage of error markers, autocompletion and quick fixes. Several frameworks and tools from different communities for creating concrete textual syntaxes for models emerged during recent years. However, these approaches failed to provide a solution in general. Open issues are incremental parsing and model updating as well as partial and federated views. Building views on abstract models is one of the key concepts of model-driven engineering. Different views help to present concepts behind a model in a way that they can be understood and edited by different stakeholders or developers in different roles. Within graphical modelling several approaches exist allowing the definition of explicit holistic, partial or combined graphical views for models. On the other hand several frameworks that provide textual editing support for models have been presented over recent years. However, the combination of both principals, meaning textual, editable and decorating views is lacking in all of these approaches. In this presentation, we show FURCAS (Framework for UUID Retaining Concrete to Abstract Syntax Mappings), a textual decorator approach that allows to separately store and manage the textual concrete syntax from the actual abstract model elements. Thereby we allow to define textual views on models that may be partial and/or overlapping concerning other (graphical and/or textual) views.

1 Introduction

Textual concrete syntaxes for models are beneficial for many reasons. They foster usability and productivity because of their fast editing style, their usage of error markers, autocompletion and quick fixes. Several frameworks and tools from different communities for creating concrete textual syntaxes for models emerged during recent years. However, they fail to provide solutions for explicit, textual, editable views as well as incremental update capabilities needed within Universally Unique Ìdentifier (UUID)-based modelling environments.

It needs to be distinguished between two different flavours of the term *view*[1]. *View (I):* The definition of what elements are displayed in a certain type of view. This is also called a *view-point*. *View (II):* For the same view-point it might still be necessary to have different instances showing the abstract concept on the same level of abstraction but providing different means of presentation. Therefore, information that are only used for the presentation need to be stored and managed separately from the actual model.

Within graphical modelling several approaches have been developed that allow to define explicitly holistic, partial or combined graphical views for models. The Graphical Modelling Framework (GMF)[2], as part of the Eclipse Modelling Framework (EMF) provides means to define and generate views for arbitrary metamodels. Furthermore, these views are not view-only but rather provide functionality to edit the underlying models through its views. The information on how a specific model element is displayed in the graphical representation is done using a decorator pattern based approach. The original decorator pattern is used to non-intrusively, dynamically add functionality to a class that is wrapped by the decorator. In our concrete case the functionality consists of information that is added to a model element that describes how it is represented in a certain view.

This means that the graphical information is clearly separated from the actual model content, allowing to define different views on the same model elements. Rational Rose (a UML modelling tool) even showed this separation to the user. If a diagram element referred to a model element that was not accessible anymore, due to whatever reason, the diagram element was still shown, however with an indicating small Ⓜ attached to it.

On the other hand several frameworks that provide textual editing support for models have been presented over recent years. Frameworks such as TCS [3] or TEF [4] allow the definition of textual concrete syntaxes (CTS) for metamodels. TEF even allows the combination of textual and graphical editors [5]. By this, it allows to define languages that only partially cover a metamodel.

The combination of both principles, meaning textual, editable and decorating views is currently not supported by any of these approaches. For example, assuming there is a textual notation for UML, editing the same UML Class using both a graphical editor and a textual editor, while preserving the layout of both views is not supported. This means that it is not possible to define views that are on the one hand textual and on the other hand allow the independent storage and management of their representation.

Furthermore, textual languages in combination with modelling pose further challenges when a UUID based model repository is employed. Such repositories assign each element a UUID that remains stable across the lifetime of the element (e.g, the MOFID in MOF 1.4) [6]. This is problematic if, as it is done in most textual modelling frameworks [7], model elements are re-created upon re-parsing of the textual representation. In large scale environments with a high number of model partitions[1] and numerous connections between these partitions, such repositories become very important. In distributed development where developers of one artefact do not always know all referrers from other model partitions to a specific model element it is crucial that elements have stable IDs. Further advantages of the UUID-based approach can be found in [8]. An initial proposal on how to solve this problem has been published in [9].

[1]Physical storage units for model elements, such as the Resource concept in EMF

2 FURCAS

The framework which is presented here is called **FURCAS**[2] (**F**ramework for **U**UID-**R**etaining **C**oncrete to **A**bstract **S**yntax Mappings). Two key features distinguish FURCAS from existing text-to-model approaches. First, FURCAS uses a textual decorator approach that allows to separately store and manage the textual concrete syntax from the actual abstract model elements (*View (II)*). Through this separation it also becomes possible to define different view points (*View (I)*) on the same model element. Second, FURCAS provides support for incremental UUID-retaining text to model (parsing) as well as incremental model-to-text (pretty printing) transformations. This means FURCAS features completely bidirectional *and* incremental mapping.

2.1 Mapping Definition

Based on the TCS approach [3] for defining mappings for textual modelling languages FURCAS employs a mapping declaration format that allows to define bidirectional mappings. The basic constructs that are used in FURCAS are based on the TCS mapping language. However, several extensions have been made to enable the language for incrementality as well as other special needs that occur in large-scale modelling environments.

FURCAS allows to use OCL [11] to resolve references within a textual view. This means that OCL, as a standard language in the modelling world for specifying constraints and queries, can be used to define actions (such as type inference) as well as "refers to" relations within the mapping definition.

Listing 1.1 shows an example for the extended mapping declaration that is used in FURCAS. Here it can be seen that name based references, such as the `to` reference of `Class` or the `name` attribute of the `TypeAdapter` are resolved respectively initialized by using OCL queries.

Furthermore, the query information, which is composed of a `query` and a `where` part can is used by the generic autocompletion as well as the incremental pretty printer. The autocompletion computes a set of candidates using the query part and uses a modified, but automatically derived version of the `where` part to suggest autocompletions during editing. The incremental pretty printer uses, for cases where no text was present for a model element, the given information to inversely construct the value that would have been used to textually reference the pretty printed element. This allows to reuse the abstractly defined OCL information in all these three different contexts.

[2]"In demonology, Furcas or Forcas is a Knight of Hell, and rules 20 legions of demons. He teaches Philosophy, Astronomy (Astrology to some authors), Rhetoric, Logic, Chiromancy and Pyromancy. Furcas is depicted as a strong old man with white hair and long white beard, who rides a horse while holding a sharp weapon (pitch fork). He knows the virtues of all herbs and precious stones, can make a man witty, eloquent, invisible (invincible according to some authors), and **live long**, and can discover treasures and **recover lost things**."[10].
This actually fits pretty well with the purpose of the framework, meaning the long living UUIDs and the recovering of model element links based on them.

Listing 1.1. Example mapping definition in TCS/FURCAS

```
template data::Class main context(root)
    :   (valueType ? ”value”) ”class” name
        (isDefined(adapters) ? ”implements” adapters{separator=”,”})
        ”{” ownedSignatures
            elementsOfType{mode=property}
        ”}”
    ;
template TypeAdapter
    : to {refersTo = name,
            query = ”self.package->select(c | c.oclIsTypeOf(data::Class))”
            where=”c.name = ?”}
      {{ name = lookIn(”’From_’.concat(self.adapted.name).” +
                        ”concat(’_to_’).concat(self.to.name)”) }}
    ;
```

The mapping declaration is also stored using the FURCAS approach and is therefore considered a model. This model is then also available at runtime allowing runtime components such as the incremental parser or the autocompletion processor to access the language definition. Furthermore, using the bootstrapped version of FURCAS to edit the mapping declarations themselves enables language developers to use intelligent migration transformations for their syntaxes. Being based on model elements with UUIDs these migrations are much more safe and controllable.

2.2 Decorator Approach

The approach allows to store all information that is needed to view and edit a textual representation for a specific model, called *domain model* while minimizing the redundancy between both, the textual decorator model, called *TextBlocks Model*, and the domain model to an absolute minimum.

The editor that comes with FURCAS is based on the eclipse text editor framework. However, it does not work on a text document but rather directly on the TextBlocks model. This means that each editing actions that is performed by the user is mapped to an action that interacts with this model. This means that the TextBlocks model not only represents a textual form of the domain model but is also responsible for storing intermediate (potentially inconsistent) states during the editing process. Figure 1 shows an overview on the decorator concept. It shows that the text that is visible within an editor is just a view that is rendered using the TextBlocks model and the domain model.

The FURCAS editor furthermore provides a generic autocompletion capability. The autocompletion processor uses the mapping definition of the current language to compute the possible completions at a specific place within the editors content. Furthermore, the OCL queries that are defined within the mapping are re-written to a from that can be executed using the information that is currently available at the input to get a list of possible completions at that specific point. Further information on the view concepts of FURCAS can be found in [12].

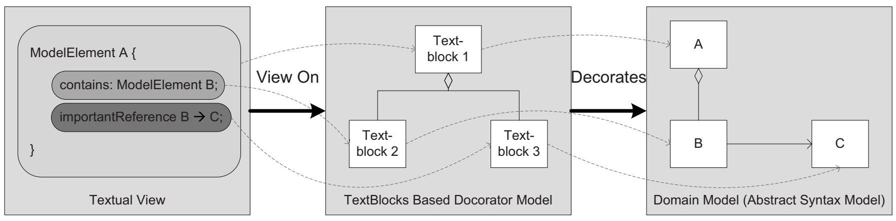

ModelElement A {
contains: ModelElement B;
importantReference B → C;
}
Textual View
View On
Text-block 1
Text-block 2
Text-block 3
TextBlocks Based Docorator Model
Decorates
A
B
C
Domain Model (Abstract Syntax Model)

Fig. 1. Overview on the decorator approach.

2.3 Transformations

FURCAS provides transformations that create an initial minimal representation for an existing domain model and transform this minimal representation into a form that is editable through our FURCAS editor.

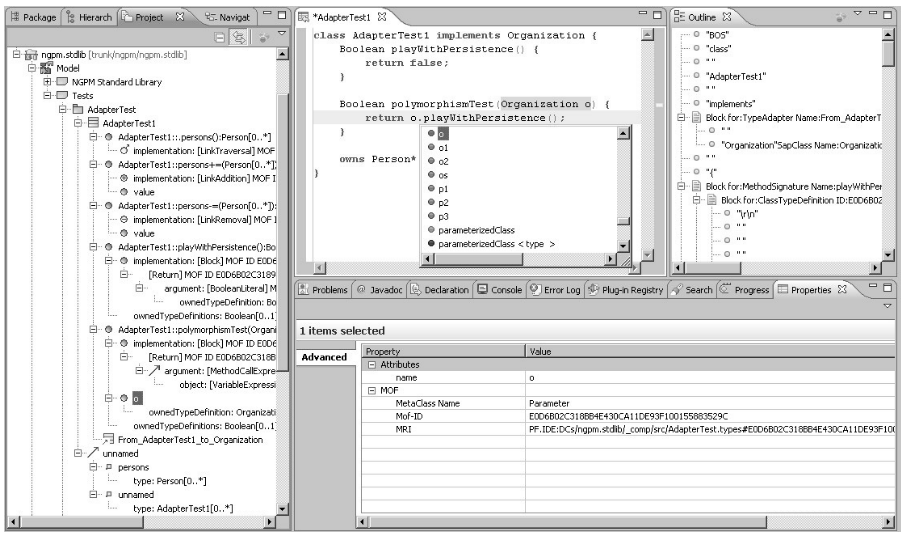
Incremental parsing techniques are used to update the corresponding domain model from intermediate editing steps. During this transformation sophisticated analysis of the existing TextBlocks model, its previous versions and its correspondencies to the domain model are performed to ensure the retainment of model elements during the process. [9] presents how incremental parsing and editing on a model level is tackled.

Changes to the domain model that are made through another view can be re-synchronized with the textual view using incremental pretty printing. During this process the current version of the TextBlocks model is checked against the changed domain model. This process is in a way different than common unparsing/ pretty-printing algorithms as existing links from the TextBlocks model to the domain model still need to be retained and maintained to ensure the TextBlocks model still serves as a consistent basis for the incremental parsing.

3 Example: Type Adapter for a Declaration Language

In this section we provide an example to illustrate the concepts of our approach. The example is based on the real-world case study that we are currently doing in cooperation with SAP AG to evaluate the overall FURCAS approach. However, to keep it understandable we only use an excerpt of the full language for illustration purposes. Figure 3 shows an example for the use of the FURCAS editor.

For the sake of simplicity let’s say that the part of the language mentioned here is quite similar to the syntax of Java. So, it is possible to declare classes which may have methods. Furthermore, the language explicitly supports associations between classes as first level entities. An additional concept that is used in this example is that of a type adapter. A type adapter is used to provide an explicit language construct that allows to adapt two classes to each other in order to fulfil a certain conformance relation (following the standard adapter pattern). Figure 3 shows a graphical as well as a textual



```
class AdapterTest1 implements Organization {
    Boolean playWithPersistence() {
        return false;
    }

    Boolean polymorphismTest(Organization o) {
        return o.playWithPersistence();
    }

    owns Person*
}
```

1 items selected

Property	Value
Attributes	
name	o
MOF	
MetaClass Name	Parameter
Mof-ID	E0D6B02C318BB4E430CA11DE93F100155883529C
MRI	PF.IDE:DCs/ngpm.stdlib/_comp/src/AdapterTest.types#E0D6B02C318BB4E430CA11DE93F10(

Fig. 2. Example of the FURCAS editor.

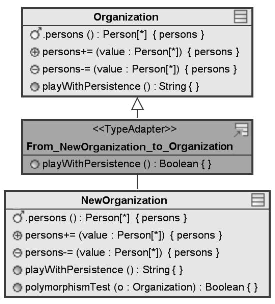
representation of a type adapter that adapts from “newOrganization” to “Organization” by providing an implementation of the method “playWithPersistence()” with an adapted return type from “String” to “Boolean”.

FURCAS is used to generate an editor that allows us to create `TypeAdapters` textually. An example for this representation can be seen in Figure 3. Furthermore, the adapter can be edited in a special view to edit the adapter methods separately. So there exists one graphical and two different textual syntaxes for the type adapter which allow the editing of the same adapter model element.

4 Conclusion

FURCAS is a framework that allows to do textual modelling a view based decorator approach. It facilitates incremental parsing and pretty printing allowing the retainment of model elements in UUID-based modelling environments. We are currently implementing a version of FURCAS that can be used on top of the Eclipse Modelling Framework. Further information can be found at `http://www.furcas.org`.

Future work will deal with improving the incrementality of the approach as well as the evolution of the mapping declarations.


Organization
.persons () : Person[*] { persons }
persons+= (value : Person[*]) { persons }
persons-= (value : Person[*]) { persons }
playWithPersistence () : String { }
<<TypeAdapter>>
From_NewOrganization_to_Organization
playWithPersistence () : Boolean { }
NewOrganization
.persons () : Person[*] { persons }
persons+= (value : Person[*]) { persons }
persons-= (value : Person[*]) { persons }
playWithPersistence () : String { }
polymorphismTest (o : Organization) : Boolean { }

```
class NewOrganization implements Organization {
    String playWithPersistence() {
        return "false";
    }

    Boolean polymorphismTest(Organization o) {
        return o.playWithPersistence();
    }

    owns Person* persons { ., +=, -= }
}
```

```
adapter From_NewOrganization_to_Organization from NewOrganization to Organization {
    Boolean playWithPersistence() {
        return NewOrganization.playWithPersistence().parseBoolean();
    }
}
```

Fig. 3. Graphical and textual representations of a Type Adapter.

References

1. IEEE: Std 1471:2000 – Recommended practice for architectural description of software intensive systems. (2000)
2. Eclipse Foundation: Graphical Modeling Framework Homepage. `http://www.eclipse.org/gmf/` (2007) Last retrieved 2008-07-06.
3. Jouault, F., Bézivin, J., Kurtev, I.: TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In: GPCE '06, New York, NY, USA, ACM Press (2006) 249–254
4. Scheidgen, M.: Textual editing framework. `http://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tool.html` (2007) Last retrieved 2008-07-06.
5. Scheidgen, M.: Textual modelling embedded into graphical modelling. In: ECMDA-FA. (2008) 153–168
6. Uhl, A.: Model-driven development in the enterprise. IEEE Software **25**(1) (2008) 46–49
7. Goldschmidt, T., Becker, S., Uhl, A.: Classification of Concrete Textual Syntax Mapping Approaches. In: Proc. of the 4th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA 2008). (2008) 169–184
8. Uhl, A.: Model-driven development in the enterprise. `https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/7237` (2007) Last retrieved 2008-07-06.
9. Goldschmidt, T.: Towards an incremental update approach for concrete textual syntaxes for UUID-based model repositories. In Gasevic, D., Lämmel, R., Wyk, E.v., eds.: 1st International Conference on Software Language Engineering (SLE). Volume 5452 of Lecture Notes in Computer Science., Springer-Verlag, Berlin, Germany (2008) 168–177
10. Mathers, S.L.M., Crowley, A.: The Goetia: The Lesser Key of Solomon the King. (1904; 1995 reprint)
11. Object Management Group: Object Constraint Language (OCL) Specification Version 2.0. OMG Document No 05-06-06
12. Goldschmidt, T., Becker, S., Uhl, A.: Textual views in model driven engineering. In: Proceedings of the 35th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), IEEE (2009)