D1.3

Version: 2.0
Date: 2010-12-23
Dissemination status: PU
Document reference: D1.3

MDSD software framework for business compliance – Final version

Project acronym:	COMPAS
Project name:	Compliance-driven Models, Languages, and Architectures for Services
Call and Contract:	FP7-ICT-2007-1
Grant agreement no.:	215175
Project Duration:	01.02.2008 – 28.02.2011 (36 months)
Co-ordinator:	TUV Vienna University of Technology (AT)
Partners:	CWI Stichting Centrum voor Wiskunde en Informatica (NL)
	UCBL Université Claude Bernard Lyon 1 (FR)
	USTUTT Universitaet Stuttgart (DE)
	TILBURG UNIVERSITY Stichting Katholieke Universiteit Brabant (NL)
	UNITN Universita degli Studi di Trento (IT)
	TARC-PL Telcordia Poland (PL)
	THALES Thales Services SAS (FR)
	PWC Pricewaterhousecoopers Accountants N.V. (NL)

This project is supported by funding from the Information Society Technologies Programme under the 7th Research Framework Programme of the European Union.

Project no. 215175
COMPAS
Compliance-driven Models, Languages, and Architectures for Services
Specific Targeted Research Project
Information Society Technologies
Start date of project: 2008-02-01 Duration: 36 months

D1.3 MDSD software framework for business compliance

Revision 2.0

Due date of deliverable: 2010-12-31
Actual submission date: 2010-12-30

Organisation name of lead partner for this deliverable:
TUV Technische Universität Wien, AT
Contributing partner(s):
CWI Stichting Centrum voor Wiskunde en Informatica, NL
TARC-PL Telcordia Poland, PL
THALES, Thales Services SAS, FR
TILBURG UNIVERSITY, Stichting Katholieke Universiteit Brabant, NL
UNITN Universita degli Studi di Trento, IT
USTUTT Universitaet Stuttgart, DE

Project funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

History chart

Issue	Date	Changed page(s)	Cause of change	Implemented by
0.1	2008-10-30	Table of contents	New document	TUV
0.2	2009-11-10	Whole document	Add content	TUV
0.3	2009-12-10	Whole document	Feedbacks from discussions with partners	TUV + partners
1.0	2009-12-23		Approve & Release	TUV
1.1	2010-08-23	Section 2 & 3	Initiate an incremental version of D1.3	TUV
1.2	2010-10-21	Section 1.2, 1.3 & 1.4	Update due to the migration of oAW to EMF	TUV
1.3	2010-11-06	Section 1.5	Add the installation guidance	TUV
1.4	2010-11-21	Section 1	Revise VbMF guidance	TUV
1.5	2010-11-29	Section 1.4, 1.5.1, 1.5.2, 1.5.3	Add one section and revise to explain VbMF preference configuration and the starting example	TUV
1.6	2010-11-29	Section 1.5, 1.6, 2.1, 2.2	Add one new section (1.6) and update others	TUV
1.7	2010-12-20	Section 5	Section on DSL Tooling	TUV
1.9	2010-12-23	Whole document	Addressing USTUTT's comments and finalization	TUV
2.0	2010-12-30		Approval	TUV

Authorisation

No.	Action	Company/Name	Date
1	Prepared	TUV	2010-12-30
2	Approved	TUV	2010-12-30
3	Released	TUV	2010-12-30

Contents

List of figures

Listings

Abstract

Ensuring business compliance in process-driven SOAs is a tedious and error-prone task because the stakeholders confront two challenges: the increasing complexity of process descriptions and the gap between abstraction levels due to the difference of language syntax and semantics, the difference of granularity, and the lack of supporting links between high-level and low-level process languages. In this deliverable, we present a MDSD software framework comprising a view-based, model-driven approach and domain-specific languages (DSL) to address the aforementioned issues in order to ease the process development in a flexible, extensible manner. The MDSD software framework provides a foundation for modelling and developing processes and business compliance as well as appropriate means for integrating COMPAS partners' efforts at both conceptual and technical levels. Furthermore, we present a DSL for specifying Quality of Service (QoS) compliance concerns that is tailored for technical and non-technical stakeholders.

1. Introduction

1.1 Purpose and Scope

Within the overall scope of the COMPAS project, WP1 focuses on development of the core and modelling aspects of this business compliance framework as well as ensuring that it integrates with the components from the different WPs in COMPAS. The purpose of this deliverable is to elaborate the MDSD software framework – which is the foundation for modelling and development of business processes and compliance. In addition, we illustrate our developed domain-specific language (DSL) for specifying the services' QoS compliance concerns in technical and non-technical manners.

1.2 Document Overview

In Section 2, we illustrate how the presented tools of this deliverable fit into the overall COMPAS architecture. Section 3 concentrates on the view-based MDSD software framework for modelling and developing business processes. In Section 4 we demonstrate our developed DSL for specifying the services' QoS compliance concerns.

1.3 Definitions and Glossary

The most important terminology concerning the COMPAS project is listed on the public COMPAS web site [D7.1] available at http://www.compas-ict.eu
The following definitions are valid only in the scope of this deliverable.

Separation of concerns: the process of breaking a software system into distinct pieces such that the overlaps between those pieces are as little as possible, in order to make it easier to understand, design, develop, maintain, etc., the system.

Domain-Specific Languages (DSLs): DSLs are small languages that are specifically tailored for a particular domain. Usually, DSLs are simple because they are suited for a very narrow purpose only, and they are easy to edit and to translate. To describe a broad domain, a broad DSL can be used. To keep the smallness and simplicity of DSLs, multiple narrow DSLs should be used, which have to be

combined to describe a broad domain. The goal of DSLs is to improve productivity and software quality. DSLs raise the level of abstraction to empower users with the ability to build solutions using concepts that are similar to the domain and his/her knowledge.

Model-Driven Software Development (MDSD) or *Model-Driven Development (*MDD*)*: a paradigm that advocates the concept of models, that is, models will be the most important development artefacts at the centre of developers' attention. In MDSD, domain-specific languages are often used to create models that capture domain abstraction, express application structure or behaviour in an efficient and domain-specific way. These models are subsequently transformed into executable code by a sequence of model transformations.

Model transformation: a transformation maps high-level models into low-level models (aka model-to-model transformations), or maps models into source code, executable code (aka model-to-code or code generation).

Stakeholders: In general, stakeholder is a person or organization with a legitimate interest in a given situation, action or enterprise. In the context of this chapter, stakeholder is a person who involved in the business process development at different levels of abstraction, for instance, the business experts, system analysts, IT developers, and so forth.

1.4 Abbreviations and Acronyms

BPEL	Business Process Execution Language
BPMN	Business Process Modelling Notation
BPML	Business Process Modelling Language
DSL	Domain Specific Language
EMF	Eclipse Modelling Framework
MDSD	Model-Driven Software Development
MOF	Meta-Object Facility
MORSE	Model-Aware Repository and Service-Environment
OAW	openArchitectureWare
QoS	Quality of Service
SOA	Service-Oriented Architecture
VbMF	View-based Modelling Framework
WSDL	Web services Description Language

2. Tool Overview

In this section we show the place of this deliverable's presented prototypes in the overall COMPAS architecture.

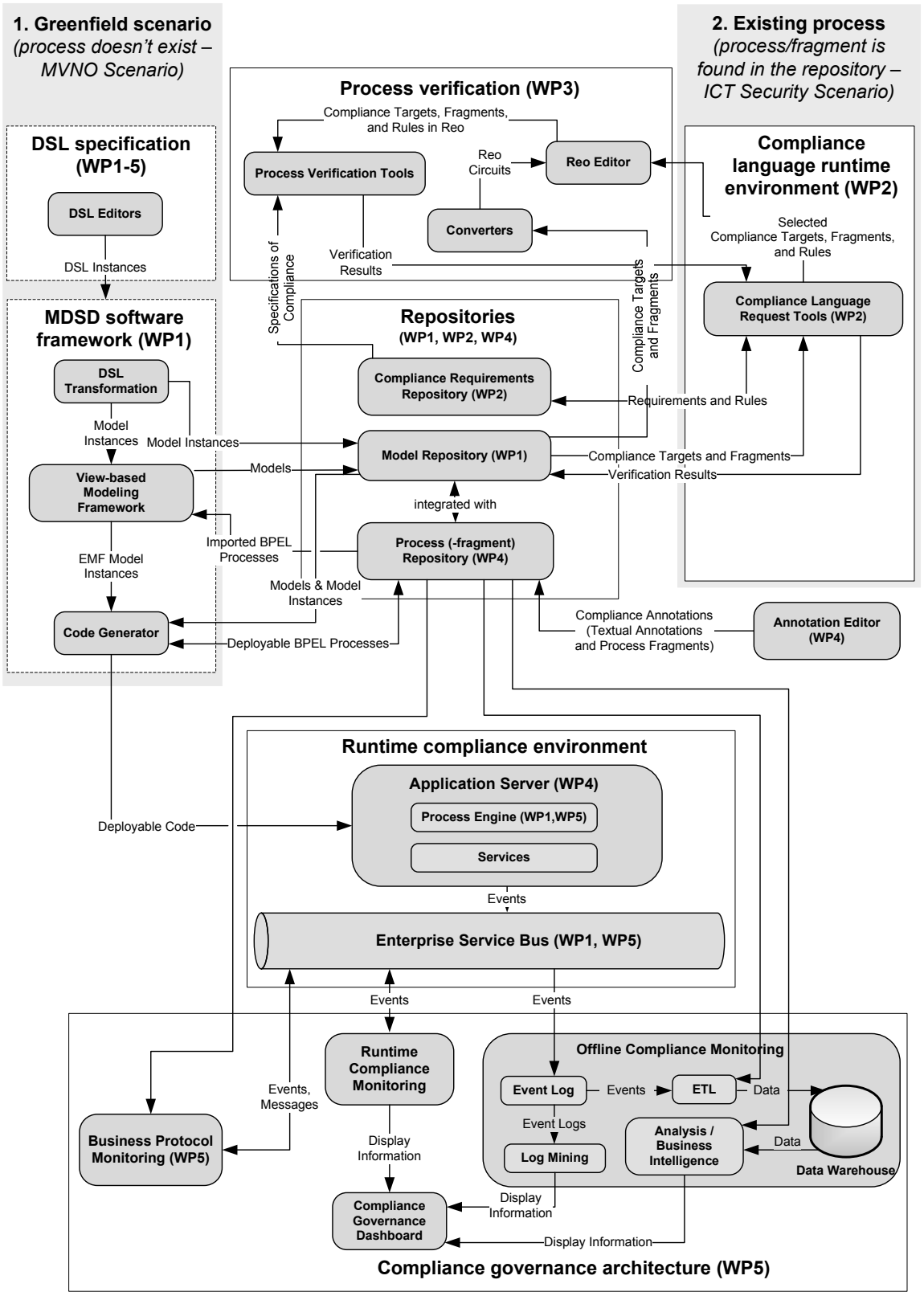


Figure 1 The COMPAS architecture

3. MDSD Software Framework prototype

In the previous release of the MDSD Software Framework, we present the implementation of our view-based, model-driven approach for process-driven SOAs in terms of a View-based Modelling Framework (VbMF). In this final release, we introduce the following additional features:

- User-friendly interface and interactions, especially for code generation, view extractions, BPEL process deployment, etc.
- Additional supports for advanced BPEL concepts such as extensions, fault handlers
- Additional supports for traceability of VbMF and BPEL elements via elements' UUID and MORSE repository meta-data.
- Integration with other prototypes: QoS DSL tooling, MORSE repository, and runtime environment.

This document accompanies the MDSD Software Framework prototype which is packaged and delivered in the corresponding COMPAS DVD at M35.

3.1. Introduction

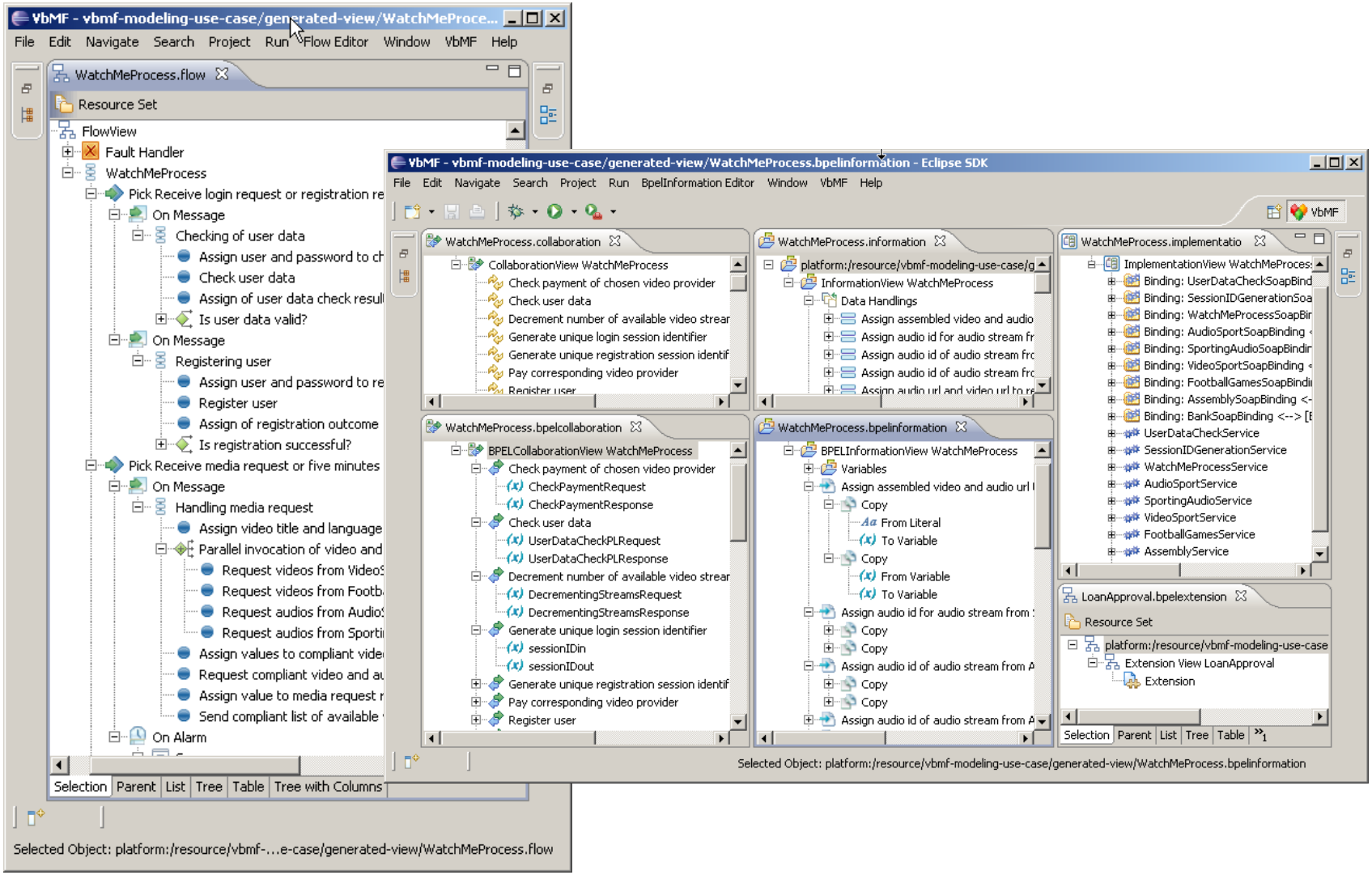
The MDSD Software Framework has been implemented based on the Eclipse Modelling Framework (EMF) and its sub-projects. One benefit of using Eclipse Modelling Framework is that we gain better integration and interoperability with existing development tools developed based on EMF Ecore, a MOF-compliant meta-model, and XMI, a standard for serializing models. The View-based Modelling Framework – part of the MDSD Software Framework – offers different editors for manipulating views, such as FlowView Editor, (BPEL)CollaborationView Editor, (BPEL)InformationView Editor, and so on.

The template-based code generation rules are developed based on the Xpand and Xtend languages provided in the Eclipse M2T project. Using these rules, our tool can automatically generate process implementations including BPEL and WSDL descriptions as well as traceability meta-data used at runtime to query information from COMPAS repositories for analysing related events and circumstances.

In the final version, we introduce several enhanced features including the view extraction, view refinement, traceability matrix for integrated with MORSE and the runtime environment, supplementary views for working with MORSE and Apache ODE engine, and many others.

3.2. View-based Modelling Framework (VbMF)

VbMF is released in terms of Eclipse plugins that provide stakeholders an integrated modelling and development environment. Figure 2 shows the high-level perspective aiming at supporting business and domain experts and a richer, low-level perspective to support IT experts in modelling and developing processes.



WatchMe FlowView **Other views of the WatchMe Process**

Figure 2 Modelling and Developing the TARC-PL Use Case using VbMF

3.3. Technology and platform dependencies

VbMF is implemented based on following major technologies and platforms:

Technology/Platform	Version	Website
Eclipse IDE Platform	3.5 or higher	http://eclipse.org
Eclipse Modelling Framework	2.4 or higher	http://eclipse.org/modeling/emf
Eclipse M2T Xpand/Xtend	0.72	http://www.eclipse.org/modeling/m2t
Apache Axis2 Framework	1.5.3	http://axis.apache.org/axis2/java/core/
Apache CXF Framework	2.3.0	http://cxf.apache.org
MORSE Repository	0.9.1	http://www.infosys.tuwien.ac.at/prototype/morse

VbMF is open source under an MIT+BSD open source license that is included in the release packages. We release the MDSD Software Framework including VbMF in terms of all-in-one Eclipse distributions that completely comprise VbMF and its dependencies. In addition, we also package the complementing technologies for process development such as the Eclipse BPEL Designer and Eclipse BPMN Modeller. The release packages can be downloaded at the following address (required COMPAs credientals)

View-based Modelling Framework

- Binary: https://www.compas-ict.eu/protected/VbMF/at.ac.tuwien.vitalab.vb-1.2.1.zip
- Source: https://www.compas-ict.eu/protected/VbMF/at.ac.tuwien.vitalab.vb-1.2.1-src.zip

All-in-one distribution

- Win32: https://www.compas-ict.eu/protected/VbMF/all-in-one/e3.5-win32-1.2.1.zip

Linux32: https://www.compas-ict.eu/protected/VbMF/all-in-one/e3.5-linux32-1.2.1.tar.bz2

3.4. MDSD Software Framework User's Guide

3.4.1. VbMF Installation

3.4.1.i. Using the all-in-one distribution

For the end-users' convenience, we release an all-in-one distribution that is a complete Eclipse 3.5.2 development environment along with the MDSD Software Framework and some complementary technologies for executable business process and Web service development such as BPEL, WSDL, XSD, XML, etc. You can download the latest version of the all-in-one distribution that is suitable to your working operating systems at https://www.compas-ict.eu/protected/VbMF/all-in-one (required a valid COMPAS credential). Then, just unzip the package and start Eclipse to work with the MDSD Software Framework.

3.4.1.ii. Manual installation

The final release of the MDSD Software Framework works fine in Eclipse 3.5 or later. In this section, we present the steps required to install the MDSD Software Framework in Eclipse 3.5.2 in Windows. These steps are rather generic and can be applied for other Eclipse distribution in other operating systems as well.

- ***Step 1***: In case you already have Eclipse 3.5.2 running, skip to step Now download the latest Eclipse 3.5.2 distribution at http://www.eclipse.org
- ***Step 2***: Unzip the Eclipse distribution into a folder, let say, *c:\eclipse*.
- ***Step 3***: Start Eclipse 3.5.2 by double clicking the file *eclipse.exe***.** In the upcoming dialog, choose a workspace to start working with.
- ***Step 4***: Install the Eclipse Modelling Framework: Choose Menu *Help* **>** *Install New Software* **...** Then pick "*Galileo* **...** " in the box "*Work with* ", open the sub-tree of "*Modeling*", then choose "*EMF - Eclipse Modelling Framework SDK*". Accept all default options in the subsequent dialogs. Waiting for the Eclipse Modelling Framework to be completely installed. After EMF is installed, you can choose "Restart Eclipse" such that the newly installed EMF components are loaded. However, it is better to close Eclipse IDE and restart it after installing VbMF plugin in Step 5.
- ***Step 5***: Download the newest binary VbMF plugin at the following address: https://www.compas-ict.eu/protected/VbMF (required a valid COMPAS credential). Copy all JAR files into the "*dropins*" folder inside the folder where Eclipse was installed, i.e., copy to *c:\eclipse\dropins*. Now, restart Eclipse IDE (see Step 3 above). You can easily check if the MDSD Software Framework has been properly installed and loaded by looking for a menu item named "*VbMF*" in the main menu.

3.4.2. Basic configuration

We demonstrate the functionality of VbMF using the Eclipse distribution and a small Greeting process. First of all, you might need to specific some basic preferences such that VbMF can work properly. Go to the menu *VbMF* and choose *Configuration*. In Figure 3 we show the VbMF's preference page. There are also other possibilities to activate the VbMF Preference page, for instance, either open the menu *Windows* to choose *Preferences*, or right click at any place in Eclipse and choose in the context menu *VbMF > Configuration*, or press shortcut *Ctrl + Shift + C*.

Error! Reference source not found.

Figure 3 VbMF's preference page

We describe the elements of the VbMF preference page in the following table.

Element	Description
MORSE base URL	The URL pointing the Model-Aware Service Environment (MORSE). As VBMF is integrated with MORSE, this URL is used for storing view models and view instances to the MORSE repository as well as retrieving, updating, removing MORSE resources (view models, views, build, etc.). This element can be left empty in case the MORSE repository

	hosted in the COMPAS server. Users can check the validity of the URL by pressing the "*Test*" button. The integration between VbMF and the runtime environment is elaborated in Section 4.1.
Apache ODE Deployment Service	The URL pointing the deployment service of the Apache ODE engine, e.g., http://localhost:8080/ode/processes/DeploymentService. As VbMF is integrated with the runtime environment [D1.4,D4.4], this URL is used by a component of VbMF that supports viewing processes deployed in an Apache ODE engine, deploying new processes or un-deploying existing processes in the ODE engine. Users can check the validity of the URL by pressing the "*Test*" button. The integration between VbMF and the runtime environment is described in Section 4.2.
Generate Traceability Meta-data	During the code generation phase, VbMF code generators can produce a traceability matrix embodied inside the BPEL process description that serves for monitoring and tracing back to the MORSE repositories at runtime. This option should be turned on for facilitating the power of model traceability at runtime and monitoring processes and compliance concerns. When this option is on, users can choose which process elements are going to be traced at runtime. The default option, as shown in 0, is to trace the following BPEL elements: variable, assign, invoke, reply, receive, correlationSet.

3.4.3. Quick start with a simple Greeting process

Hello World is a very well-known example to illustrate a certain programming language or technique. In this section, we present a similar example, namely, a Greeting process, to illustrate how VbMF works. The subsequent quick start guidance shows how to create a working *Greeting process* with VbMF.

3.4.3.i. Creating a Greeting process

The following steps can be used to quickly create an example *Greeting process* provided by VbMF (see **Error! Reference source not found.**):

- Step 1: Star Eclipse. Go to menu *File* and choose *New … Project*.
- Step 2: In the upcoming dialog for creating a new project, browse to *VbMF* and choose "*Greeting Example Project*".
- Step 3: Either enter an arbitrary name for the project or use the default name "*Greeting*". After all, click *Finish*.
- Step 4: VbMF creates a new project with a number of starting views.

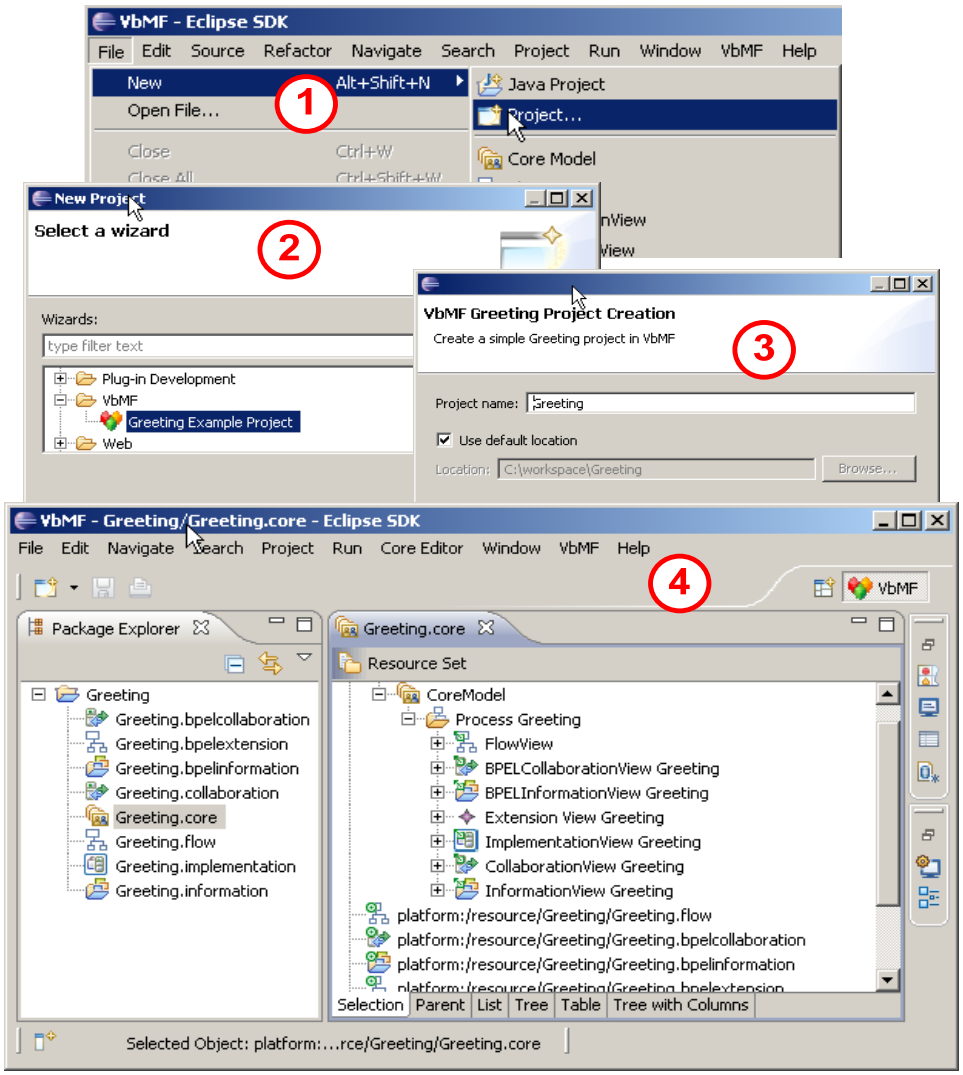


Figure 4 Step-by-step starting with the VbMF Greeting process

3.4.3.ii. Code generation

Figure 4 shows the Greeting project in the VbMF modelling and development environment after finishing all aforementioned steps. Some starting view models have been created. You can open and manipulate any of these view models. After modifying the view models, you can start generating the implementation of the Greeting process in BPEL/WSDL.

Please note that, the traceability matrix can be generated along according to the option "*Generate Traceability Meta-data*" in the VbMF preference page mentioned above. Details on the usage of the traceability matrix can be found in [D1.4, D4.4].

One can also choose the option "... *with WSDL files*" to generate all accompanying WSDL descriptions and the option "... *with deployment files*" to generate deployment configurations (*deploy.xml*) required for deploying the process to an Apache ODE process engine.

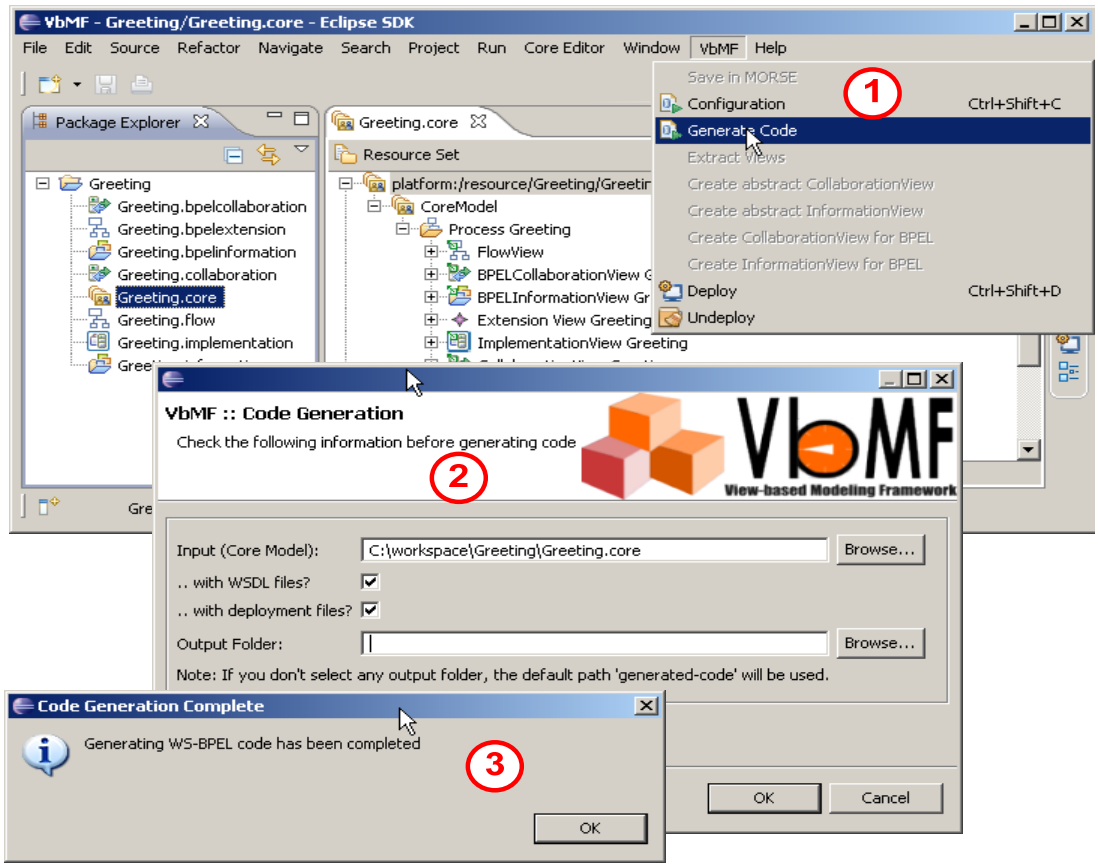


Figure 5 Generation of executable code and deployment configuration

3.4.3.iii. Greeting Process Deployment

After generating process codes and deployment configuration, as shown in Figure 5, the code can be deployed on the Apache ODE engine. Choose the menu *VbMF* and "*Deploy*" to open the VbMF deployment dialog (see Figure 6).

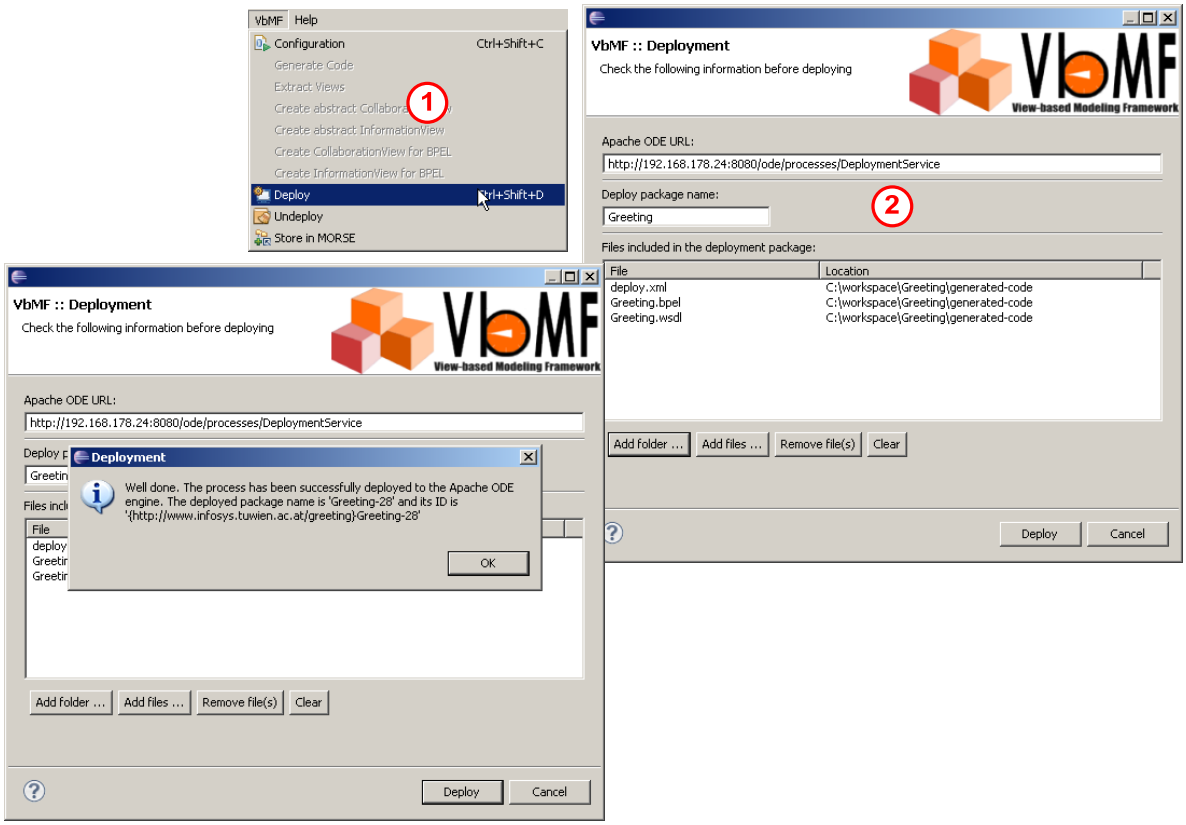


Figure 6 Deployment to the Apache ODE engine

In the VbMF deployment dialog, one can select the BPEL/WSDL description files and configuration files, such as *Greeting.BPEL*, *Greeting.wsdl*, and *deploy.xml*, generated in the previous steps for deploying. After that, a notification is display to show whether the deployment has been successfully done or not.

3.4.3.iv. View extraction

So far we have successfully modelled the Greeting process in VbMF, generated the BPEL and WSDL descriptions as well as the deployment configurations out of the view models, and deployed the Greeting process to an Apache ODE engine. However, in reality it is not often the case that the end-users always have some starting view models in hand at the beginning. One often has to invest reasonable efforts to start modelling and implementing processes from scratch. Fortunately, in case legacy process descriptions in BPEL and WSDL existed, VbMF can help the stakeholders in easing this task by providing view extraction mechanisms based on the view-based reverse engineering approach [D1.2]. In this way, one can easily achieve view models out of legacy process descriptions for many purposes such as to analyse, comprehend, or modify the process to satisfy new requirements. In the subsequent explanations, we assume that existing BPEL and WSDL descriptions of the Greeting process are there. Right click at the BPEL file, then either choose the main menu *VbMF* or the context menu "VbMF". In both cases, choose "*Extract Views*" to start view extractions (see Figure 7).

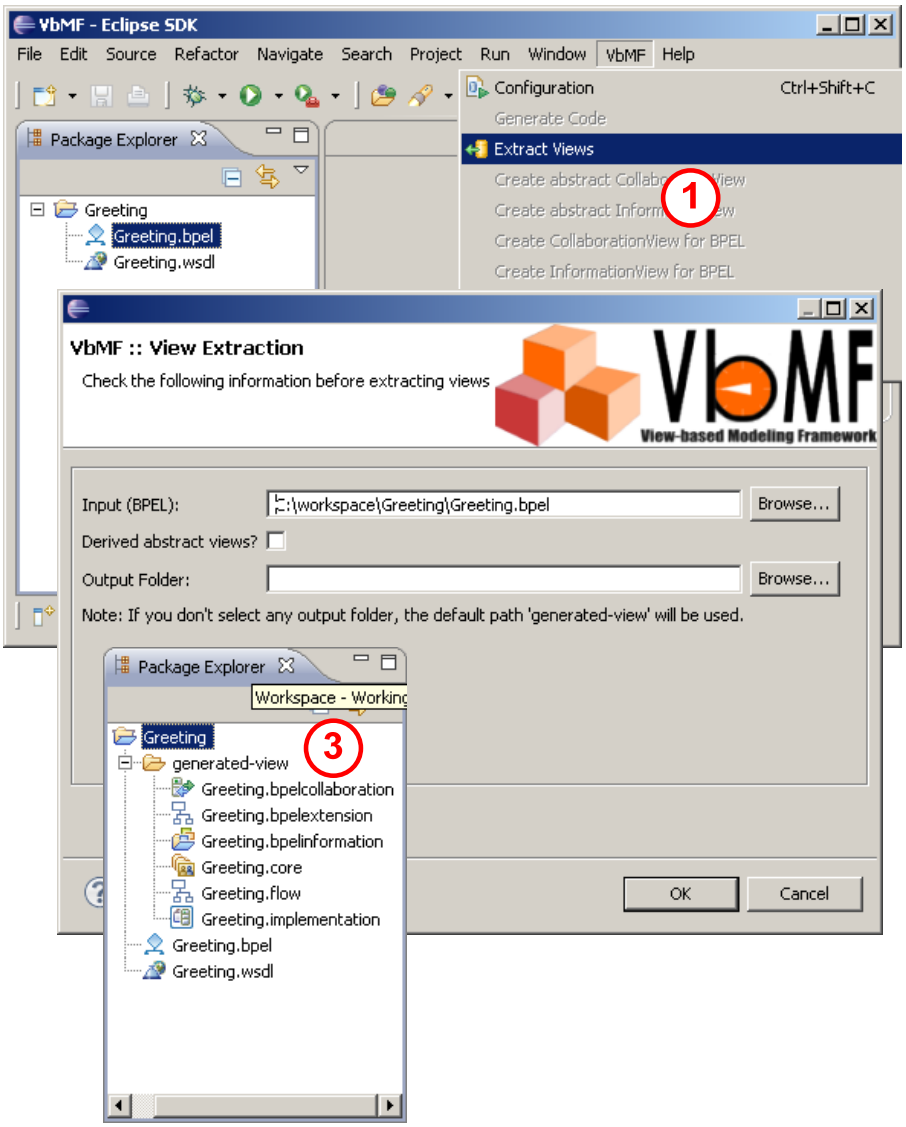


Figure 7 Extraction of view models out of existing process descriptions

VbMF view models such as FlowView and BPELCollaborationView, BPELInformation View, ImplementationView, etc., are extracted. In the dialog, one can opt to generate abstract views but this is not mandatory. The view abstraction mechanism can be used to achieve abstract views from BPEL-specific views later.

3.4.3.v. View abstraction and view refinement

Generally speaking, view abstract mechanism is used to achieve an abstract view from a corresponding technology-specific one. For instance, an abstract Collaboration view of the Greeting process can be quickly obtained if one has a BPEL-specific Collaboration view in hand (e.g., using view extractions). Right click on a BPELCollaboration view (i.e., *Greeting.BPELcollaboration*), choose menu "*VbMF*", then "*Create abstract CollaborationView*". An abstract Information view can be derived from its BPEL-specific counterpart in the same manner.

In the opposite direction, the view refinement mechanisms are very useful for producing a skeleton BPEL-specific view based on an abstract one. This mechanism is very crucial because it acts as a bridge between the business experts and IT experts. For instance, the business experts sketch out the basic behaviour and high-level business objects using VbMF abstract views. The IT experts then can facilitate the view refinement mechanism to refine those abstract views down to technology-specific layer, for instance, BPEL and WSDL in this case. Of course, the resulting views still need to be filled out with further additional information. Nevertheless they save several efforts on manually translating abstract, high-level concepts to the technical concepts and definitely become good starting points for the IT experts. A view refinement can be conducted in a similar manner as the view abstraction. Choose an abstract view, for instance, the Greeting Collaboration view (i.e., *Greeting.collaboration*), right click and choose the menu "*VbMF*", then "*Create CollaborationView for BPEL*". A Greeting BPELCollaboration view (i.e., *Greeting.BPELcollaboration*) shall be created.

3.5. MDSD Software Framework Developer's Guide

In this section, we briefly explain the source code released along with the binary distribution of the View-based Modelling Framework and provide initial guidance for developers who are interested in this framework.

3.5.1. Download source code

View-based Modelling Framework is open source under a MIT+BSD license. Its source code can be downloaded from the following site (with COMPAS credentials). The integrity of the source code can be checked again the accompanying MD5.

https://www.compas-ict.eu/protected/VbMF/at.ac.tuwien.vitalab.vb-1.2.1-src.zip
https://www.compas-ict.eu/protected/VbMF/at.ac.tuwien.vitalab.vb-1.2.1-src.zip.md5

3.5.2. Import to Eclipse

Start Eclipse and go to the main menu "*File*", "*Import*". Point to the folder where the source code has been unzipped. After the source code is successfully imported, the Eclipse project explorer looks like what is shown in Figure 8. The source code is organised into a number of Eclipse plugins that are explained in the following table.

Eclipse plugin project	Description
at.ac.tuwien.vitalab.vb	This is the base project providing some helper functions and constants.
at.ac.tuwien.vitalab.vb.model	This is the core project comprising the view models in terms of Ecore models (*.ecore) and the Java code generated by the Eclipse Modeling Framework.
at.ac.tuwien.vitalab.vb.lib	This project provides some libraries commonly used in other VbMF plugins.
at.ac.tuwien.vitalab.vb.oaw	This project implements the code generation based on the Eclipse M2T technologies (Xpand/Xtend/MWE). The code

	generation will be used in the VbMF UI project.
at.ac.tuwien.vitalab.vb.morse	This project provides the implementation of the integration between VbMF and MORSE that is used in the VbMF UI project.
at.ac.tuwien.vitalab.vb.ode	This project provides the implementation of the integration between VbMF and the runtime environment described in Section 4.2. This integration is invoked by the VbMF UI project.
at.ac.tuwien.vitalab.vb.editor	This project contains the tree-based editors generated by the Eclipse Modeling Framework that have been customized to provide the end-users with friendlier interfaces for manipulating VbMF.
at.ac.tuwien.vitalab.vb.reverse	This project implements the view extraction mechanism based on XML parsing technologies that is used by the UI project.
at.ac.tuwien.vitalab.vb.ui	This project provides the front-end that exposes all functionality of VbMF to the end-users, for instance, the tree-based view editor, the menu, the preference configuration, the MORSE and ODE views, etc.

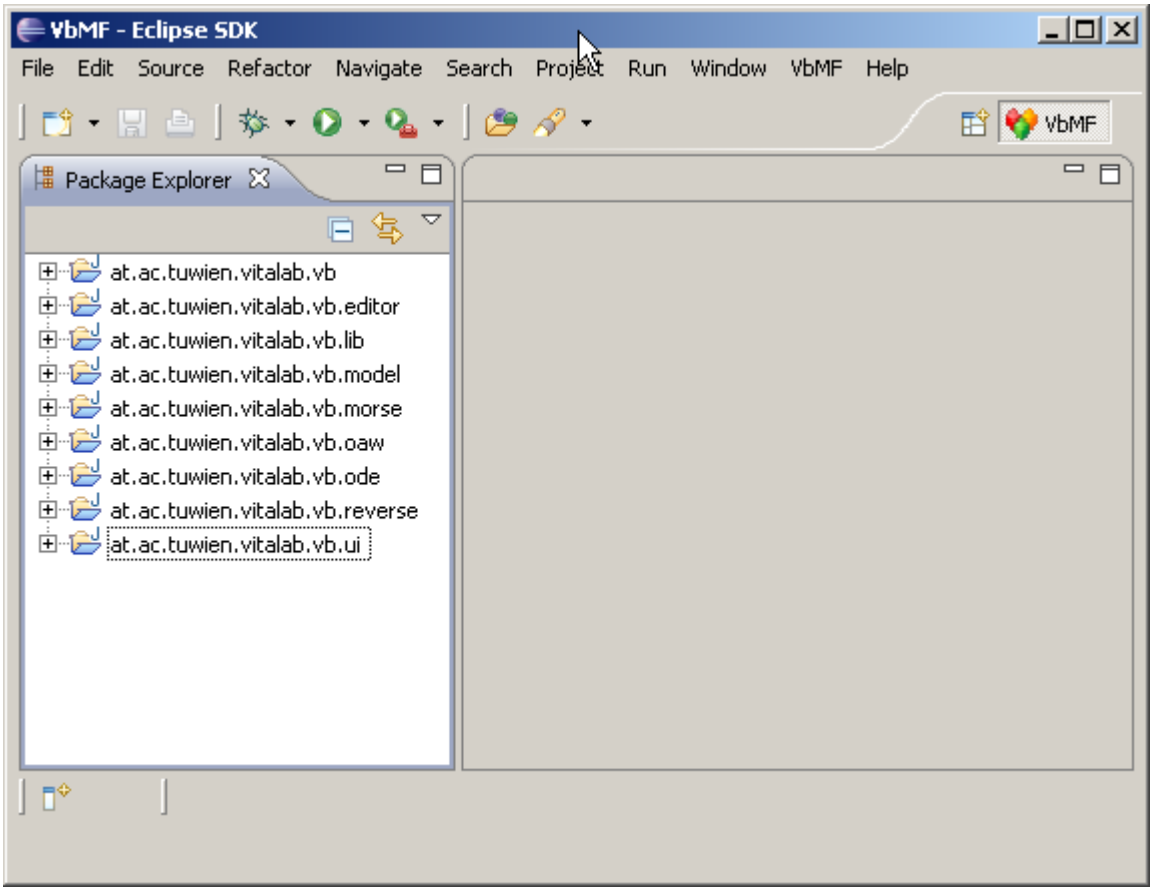


Figure 8 The Eclipse plugin development environment with VbMF source code

4. Integration between VbMF and related prototypes

4.1. Integration with the Model-Aware Service Environment

The Model-Aware Service Environment (MORSE) [D4.4] offers a model repository and model-aware services used to work with the model repository. Facilitating the Web services provided by MORSE implementation, we implemented the integration between VbMF and MORSE in which VbMF artefacts such as view models and view instances can be stored and shared via MORSE as well as checked out later.

Figure 9 shows the MORSE and MORSE Build tree views provided in the VbMF development environment for interacting with MORSE. The MORSE tree view displays the MDD resources stored including the view models and their corresponding view instances. The displayed information includes the artefact's name, creation date, UUID, and namespace URI (if any). The MORSE Build tree view is used to display the build meta-data. That is, in code generation cycle, some inputs view models are consumed and process implementation are produced. The implementation then is deployed and executed in the process engine. In order to trace back to analyse and determine the root cause of any compliance violations, we have to know the corresponding between the implementation and the view models used to generate that implementation. The build meta-data is for this purpose. In each code generation cycle, the code generator automatically creates and stores relevant build meta-data in MORSE. These meta-data can be seen in the MORSE Build tree view.

One can perform some following activities in the (1) MORSE tree view and (2) MORSE Build tree view. We annotate the number (1) and (2) to indicate that the corresponding activity is only available for the MORSE tree view and MORSE build tree view, respectively.

Activity	**Description**
Delete (1 & 2)	Right click to choose a single item or Ctrl + right click to choose multiple items, and then choose the context menu "*VbMF*" and "*Delete*". Provide your confirmation in the upcoming dialog.
Refresh (1 & 2)	Either right click and choose the context menu "*VbMF*" and "*Refresh*" or choose icon in the view toolbar menu.
Check out (1)	Right click to choose a single item or Ctrl + right click to choose multiple items, and then choose the context menu "*VbMF*" and "*Check out*". Provide the storage place in the upcoming dialog.
View (1)	Double click on any of the MORSE item
Store (1)	Choose the main or the context menu "*VbMF*", then "*Store in MORSE*". In the upcoming dialog, add multiple resources to the queue. Alternatively, one can quickly choose multiple resources (view models, view instances) in the Eclipse explorer view and right click "VbMF" "*Store in MORSE*". These resources are automatically added in to the queue. Finally, press the button "*Store*" and wait for MORSE to store the selected resources (see 0).
Search (1&2)	One can search for particular items by enter some texts in the search

	box. Figure 9 show an example of searching for items containing the text *WatchMe* in MORSE.
Expand All (1&2)	One can expand the whole tree view by click on the icon
Collapse All (1&2)	One can collapse the whole tree view by click on the icon

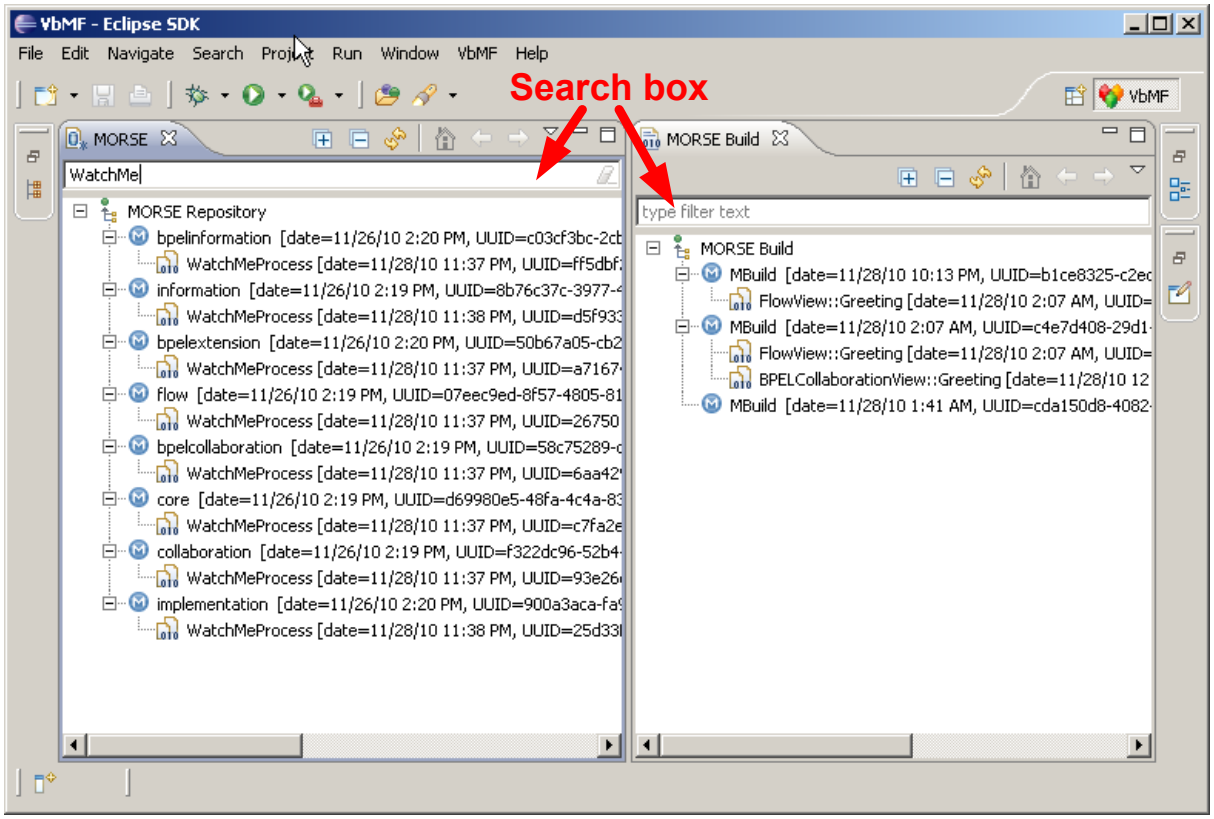


Figure 9 The MORSE and the MORSE Build tree views provided by VbMF

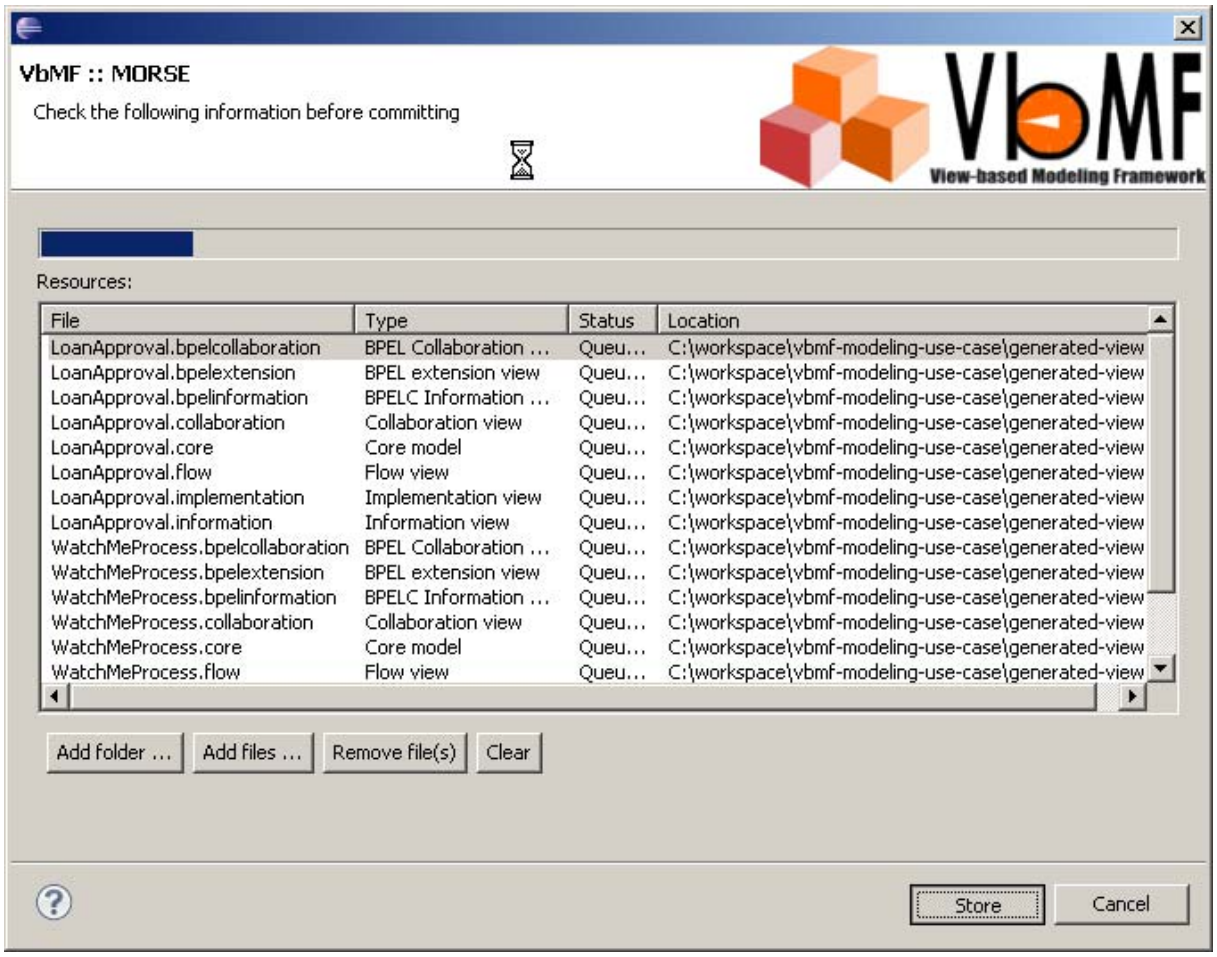


Figure 10 Storing VbMF artefacts in the MORSE repository

4.2. Integration with the runtime environment

According to the COMPAS overall architecture in [D1.1], the MDSD Software Framework, in particular VbMF, is integrated with the runtime environment in the following manners:

1. Executable BPEL and WSDL description are generated out of VbMF view models. This task has been illustrated in Section 3.4.3.ii.
2. Traceability matrix is generated based on information from the view models as well as the build meta-data.
3. Generated executable process descriptions are deployed to the BPEL engine. This task has been presented in Section 3.4.3.iii.

Concerning the second task, VbMF supports the generation of the traceability matrix [D1.4, D4.4] whenever the option "*Generate traceability meta-data*" in the VbMF preference page enabled (default). We extended the code generation and utilized the Xpath generator of the Apache ODE engine implementation in order to precisely generate the query of each traceability row as expected by the extended Apache ODE engine described in [D1.4, D4.4]. According to specific needs, one can choose which BPEL elements must be traced.

Regarding the third task, VbMF provides an ODE tree view that displays the BPEL processes deployed in a certain Apache ODE engine. One can easily switch the option "*Apache ODE Deployment Service*" in the VbMF preference page to point to an arbitrary Apache ODE engine. As the deployment step has been presented in Section 3.4.3.iii, we explain in this section the ODE tree view. In order to open this view, go to main Eclipse menu "*Window*" and "*Show View*" and choose "*ODE*". The ODE view connects to the Apache ODE engine specified by the option "*Apache ODE Deployment Service*" in the VbMF preference page to retrieve the information about the deployed processes and displays the information in the view (see 0). The basic activities can be performed with the ODE view. The information will be refreshed by clicking the icon. To un-deploy a certain process, right-click the process and choose "*Undeploy*".

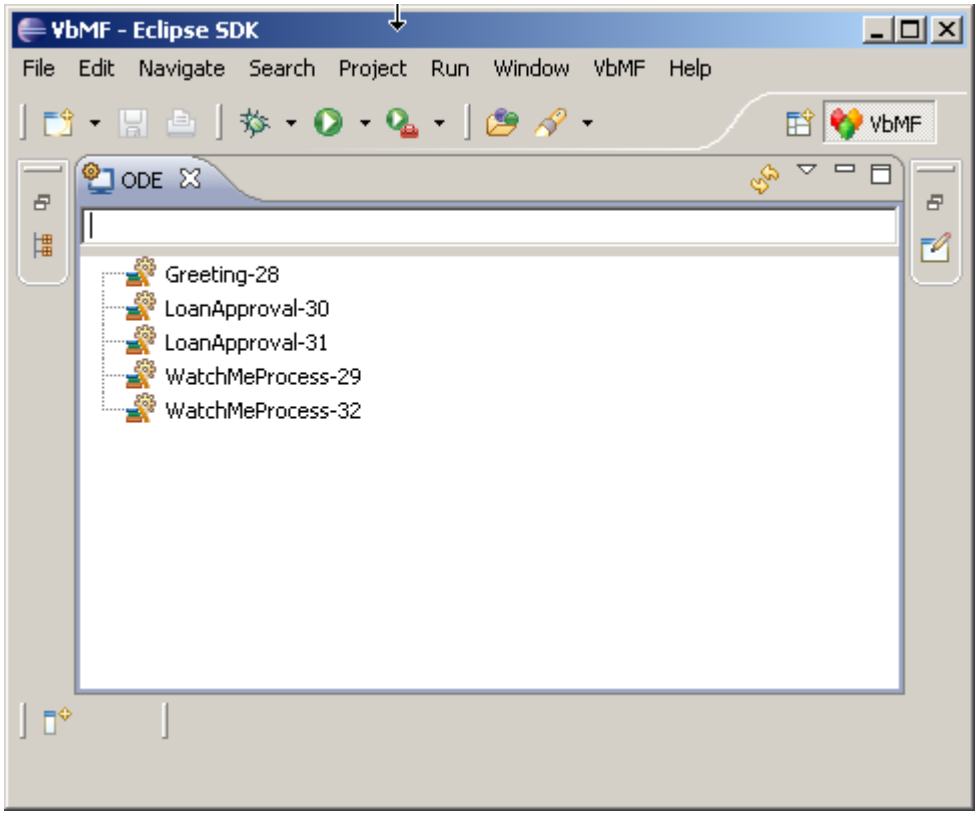


Figure 11 The ODE tree view

For massively un-deploying multiple processes, go to the main or context menu "*VbMF*" and "*Undeploy*" to open the "*Undeployment dialog*". Press the button "Refresh" to show information about the deployed process (which is similar to that of the ODE view). Then one can choose multiple processes to be un-deployed or simply press "*Undeploy All*" to un-deploy all processes. *Caution*: These tasks cannot be undone.

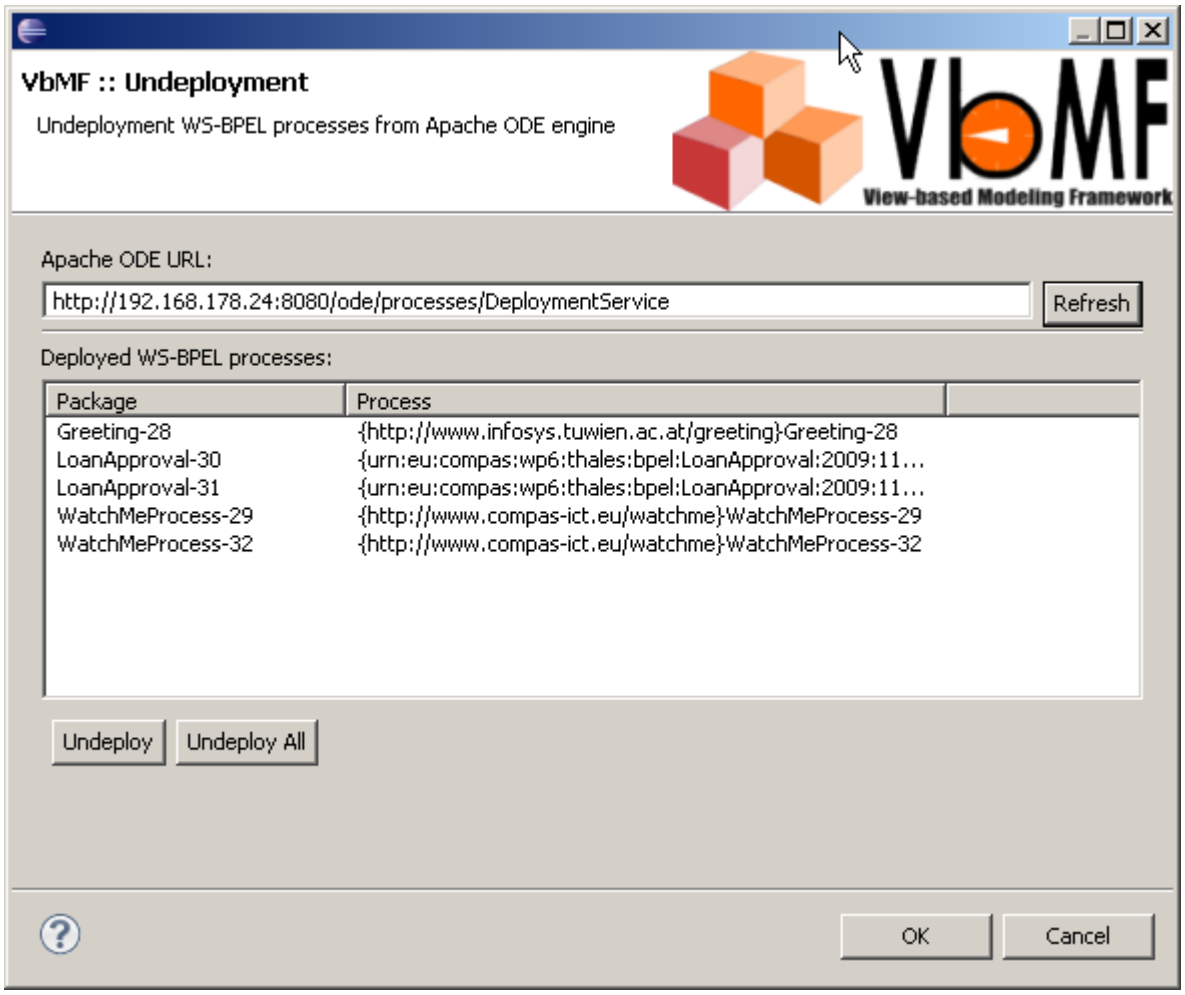


Figure 12 Massive un-deployment processes

5. Quality of Service Language (QuaLa)

In this section we present our developed Quality of Service Language (QuaLa). The language provides the facilities for describing Service Level Agreements (SLA) that contain negotiations between service provider and service customer regarding the services' quality. QuaLa concentrates especially on the specification of performance-related Quality of Service (QoS) properties, such as processing time, or availability. We separate QuaLa into two sub-languages. The high-level QuaLa is tailored for experts of the QoS domain and provides constructs for specifying the services' runtime QoS constraints that should not be violated during the system's runtime. The high-level QoS constraints need additional technical details, so that we can generate the services and the QoS monitoring infrastructure automatically. The low-level QuaLa is tailored for technical experts for specifying the additionally required technical details about how and where to monitor the required QoS properties. Hence, the low-level QuaLa extends the high-level QuaLa.

In the following we provide screenshots about using the high- and low-level QuaLa, how to merge the high- and low-level specifications, and which code is generated.

5.1.1. QuaLa's Used Technologies

We develop QuaLa using the following technologies:

Technology/Platform	Version	Website
Java	1.5.0	http://www.java.com
Frag	0.90	http://frag.sourceforge.net
Apache CXF Framework	2.3.0	http://cxf.apache.org
Apache Ant	1.8.1	http://ant.apache.org

5.1.2. The High-Level QuaLa

The high-level QuaLa is an external DSL [Fow10] that is tailored for domain experts. Hence, the high-level QuaLa's concrete syntax is different from the language in which we implemented it, i.e., Frag[1].
In Listing 1 we illustrate the QuaLa's concrete syntax using the Extended Backus-Naur Form (EBNF).

```
sla-specification = sla-name '{' [service-name '{' qos-constraints '}']* }
qos-constraints = rule '=>' action
rule = constraint [logical-operator ['(']? Constraint [')']?]
constraint = qos-property operator predicate
qos-property = 'Availability' | 'ProcessingTime' |
               'DeliveryRate' | 'MinimalFrameRate'
operator = '<' | '<=' | '>' | '>='
predicate = number unit
unit = '%' | 'd' | 'h' | 'm' | 's' | 'fps'
logical-operator = 'AND' | 'OR'
action = mail-action | sms-action
mail-action = 'mailto' '"' mail-address '"'
sms-action = 'smsto' '"' phone-number '"'
```

Listing 1 The high-level QuaLa's concrete syntax

For a better understanding of the high-level QuaLa, we give an example of the WatchMe case study in Listing 2.

```
WatchmeSLA {

   Login {
       Availability>99% => mailto "sysadmin@mvno.org",
       ProcessingTime<30sec => smsto "+12 34 56789"
   }

   Search {
       Availability>99% AND ProcessingTime<2min
         => mailto "sysadmin@mvno.org"
   }

   Stream {
       MinimalFrameRate>30fps => mailto "sysadmin@mvno.org",
       DeliveryRate>80% => smsto "+12 34567 89"
   }
}
```

[1] http://frag.sourceforge.net

Listing 2 Using the high-level QuaLa in the WatchMe case study

In this example, we specify the QoS compliance concerns of the three WatchMe services. For example, the UserDataCheck web service should have an Availability>99%. If it's lower, then send an email to the system administrator. Also, the UserDataCheck web service should have a ProcessingTime<30sec. If it's higher, then send an SMS to the specified number.

5.1.3. The Low-Level QuaLa

The low-level QuaLa is an embedded DSL [Fow10]. An embedded DSL's concrete syntax is equivalent to the concrete syntax of the language workbench with which it was implemented. In our case the low-level QuaLa's concrete syntax is equivalent to Frag's syntax [Zdu09]. For further information about Frag please refer to: http://frag.sourceforge.net

5.1.3.i. Specifying the Technology's Architecture

In our work, we implement the WatchMe web services using the Apache CXF web service framework[2]. Hence, the technical experts have to specify the architecture of the Apache CXF web service framework. The message-flow between client and server as based on so called chains, and each chain consists of multiple phases. We hook QoS interceptors [OZD10] for measuring the web services' QoS compliance concerns.
The low-level QuaLa is an embedded DSL [Fow10]. Its concrete syntax is equivalent to the language in which we implemented it, i.e., Frag. In Listing 3 we illustrate how the technical experts specify the CXF architecture and where to place the interceptors for measuring the QoS compliance concerns.

```
## CHAIN ##
cxf::InChain create ServerIn
## PHASES ##
cxf::InPhase create InPreInvoke
cxf::InPhase create InInvoke
## assign phases to chain ##
ServerIn phases {InPreInvoke InInvoke}

## PROCESSING TIME ##
ProcessingTime classes cxf::QoS
ProcessingTime chains ServerIn
ProcessingTime phases {InPreInvoke InInvoke}
```

Listing 3 Using the low-level QuaLa to specify the CXF architecture

In Listing 3 we illustrate how technical experts have to specify the CXF architecture. First, we define the chains and phases of the CXF web service framework. Then, we assign the phases to the chains. Afterwards we define in which phases of which chain the QoS properties have to be measured. In our example, we define that the processing-time is measured in the `IN-Chain` of the server between the `Pre-Invoke` and `Invoke` phases.
The architecture of the used web service framework has to be defined only once, because the QoS properties will be measured for each service invocation in the same phases of the same chain. Changing the architecture implies to change the architecture description and the low-level QuaLa's language model.

[2] http://cxf.apache.org

5.1.3.ii. Extending the High-Level Service Specifications with Technical Details

Another utilization of the low-level QuaLa is the specification of technical details of the high-level service definition. In Listing 4 we illustrate how to add technical concerns to high-level services `Login`, `Search`, and `Stream`.

```
## LOGIN SERVICE
Login classes cxf::Service
Login package "eu.compas.watchme"
Login uri "http://localhost:5001/watchme/login"
Login wsdl "http://localhost:5001/watchme/login?wsdl"
Login namespace "http://www.compas-ict.eu/watchme/login"
Login operations [list build \
  [cxf::Operation create login -name "login" -returnType UUID -parameters
[list build \
    [cxf::Parameter create pUsername -name "username" -type String]
    [cxf::Parameter create pPassword -name "password" -type String]]]]

## SEARCH SERVICE
Search classes cxf::Service
Search package "eu.compas.watchme"
Search uri "http://localhost:5001/watchme/search"
Search wsdl "http://localhost:5001/watchme/search?wsdl"
Search namespace "http://www.compas-ict.eu/watchme/search"
Search operations [list build \
  [cxf::Operation create search -name "search" -returnType String -
parameters [list build \
    [cxf::Parameter create sMovie -name "movie" -type String]
    [cxf::Parameter create sLanguage -name "language" -type String]]]]

## STREAM SERVICE
Stream classes cxf::Service
Stream package "eu.compas.watchme"
Stream uri "http://localhost:5001/watchme/stream"
Stream wsdl "http://localhost:5001/watchme/stream?wsdl"
Stream namespace "http://www.compas-ict.eu/watchme/stream"
Stream operations [list build \
  [cxf::Operation create stream -name "stream" -parameters [list build \
    [cxf::Parameter create sStreamID -name "streamID" -type String]]]]
```

Listing 4 Specifying the web services' technical details using the low-level QuaLa

We define for each service – `Login`, `Search`, and `Stream` – a package, an URI, to location of the WSDL-file, a namespace, an its operations. For example, the `Login` web service has a `login` operation that takes a username and a password as input.

5.1.4. Code Generator

To get a running system out of the QuaLa specifications, the QuaLa consists of a code generator that produces executable code. The code generator generates the following components:

- A skeleton of the web service implementation that must be extended with the web service's behaviour manually
- The interceptors for measuring the QoS properties
- A host that hosts the web services
- A client that can be used for testing reasons

In our work, we use a template-based code generation approach. In the following we demonstrate how the code generator for the interceptors that measure the QoS properties works.

```
public class <~GeneratorUtils removeNamespace [$condition qos]~>Interceptor
   extends AbstractSoapInterceptor {

...

 public void handleMessage(SoapMessage msg) throws Fault {
<~ self applyIf {[list length [[$condition qos] phases]] == 2} {

  if(this.getPhase().equalsIgnoreCase(Phase.
   <~QuaLaGenerator mapPhase [list index [[$condition qos] phases] 0]~>)) {

      InvocationContext qos =
          (InvocationContext)msg.get(InvocationContext.class);
      if(qos==null) {
         qos = new InvocationContext();
      }

      qos.set
       <~GeneratorUtils removeNamespace [$condition qos]~>(
            System.nanoTime());
      msg.setContent(InvocationContext.class, qos);
  } else if(this.getPhase().equalsIgnoreCase(Phase.
   <~QuaLaGenerator mapPhase [list index [[$condition qos] phases] 1]~>)) {

      InvocationContext qos=
         (InvocationContext)msg.getContent(InvocationContext.class);
    if(qos!=null) {
     long nDiff = System.nanoTime()-qos.get
          <~GeneratorUtils removeNamespace [$condition qos]~>();
     System.out.println("
        <~GeneratorUtils removeNamespace [$condition qos]~>
            Interceptor => "+nDiff);
     qos.set<~GeneratorUtils removeNamespace [$condition qos]~>(nDiff);
    } else {
     throw new Fault(
       new Exception("
         <~GeneratorUtils removeNamespace [$condition qos]~>
           not found in invocation context!"));
    }
  }
 }

...
}
```

Listing 5 Code generation template for an interceptor

In Listing 5 we illustrate the template for generating an interceptor that measures the QoS properties. As the QoS properties are measured between two phases of the message-flow, interceptors are placed in these two phases. In the first phase, the QoS interceptor takes the current time and puts it into the invocation context. In the second phase, the interceptor calculates the elapsed time, i.e., the time difference between the first execution of the interceptor and the current time. The calculated time difference is again stored in the invocation context and can be used for further processing.

5.1.5. Generated Code

In Listing 6 we demonstrate the generated interceptor for measuring the processing time. As specified in the low-level QuaLa (see Listing 3), the interceptor is placed in two phases – `PreInvoke` and `Invoke`. As specified in the code generation template (see Listing 5), in the PreInvoke phase the interceptors puts the current time into the invocation context, and in the Invoke phase the interceptors calculates the elapsed time, i.e., the processing time.

```
public class ProcessingTimeInterceptor extends AbstractSoapInterceptor {

...

 public void handleMessage(SoapMessage msg) throws Fault {
  if(this.getPhase().equalsIgnoreCase(Phase.PRE_INVOKE)) {
   InvocationContext qos =
       (InvocationContext)msg.get(InvocationContext.class);
   if(qos==null) {
    qos = new InvocationContext();
   }
   qos.setProcessingTime(System.nanoTime());
   msg.setContent(InvocationContext.class, qos);
  } else if(this.getPhase().equalsIgnoreCase(Phase.INVOKE)) {
   InvocationContext
      qos=(InvocationContext)msg.getContent(InvocationContext.class);
   if(qos!=null) {
    long nDiff = System.nanoTime()-qos.getProcessingTime();
    System.out.println("ProcessingTimeInterceptor => "+nDiff);
    qos.setProcessingTime(nDiff);
   } else {
    throw new Fault(
    new Exception("ProcessingTime not found in invocation context!"));
   }
  }
 }

...

}
```

Listing 6 The generated interceptor for measuring the processing time

5.1.6. Using QuaLa in the WatchMe Case Study

First, download the QuaLa prototype at: http://www.infosys.tuwien.ac.at/prototype/QuaLa/ Then, unzip the `quala.zip` file using your favourite unzipping tool.

Within the prototype, we provide the high- and low-level specifications of the WatchMe case study's QoS compliance concerns. We placed the high-level specifications into:

```
/path/to/quala/case-study/watchme/watchme-high-level.sla
```

The low-level specifications are located in the file

```
/path/to/quala/case-study/watchme/watchme-low-level.frag
```

For starting the QuaLa code generator, execute the following command:

```
/path/to/quala> ant quala
        -Dhigh-level="./case-study/watchme/watchme-high-level.sla"
        -Dlow-level="./case-study/watchme/watchme-low-level.frag"
```

Within a short time, the DSL's code generator generates the WatchMe web services and the interceptors that measure the QoS properties. The generated code is placed in the `src-gen` folder. In the `src` folder, our code generator generates the skeletons for the web service implementations that have to be extended manually with the web services' behaviour. For example, the developer has to implemented the behaviour of the Login web service in:
`src/eu/compas/watchme/Login.java`

Then, one can start the services by starting the service host, using Apache Ant[3]. Just execute the following command:

```
/path/to/quala> ant run
```

6. Conclusion

In this deliverable we present the D1.3 prototype's final version. We enhanced the tool support for the View-based Modelling Framework and DSL tooling [D1.2] smoothly and completely integrated our tools with the Model-Aware Service Environment (MORSE) and runtime-supporting infrastructure [D4.2, D4.4].

7. Reference documents

7.1. Internal documents

[D1.1] "Model-driven Integration Architecture for Compliance", final version of 2009-06-08.
[D1.2] "Core meta-models, transformation templates, and languages", 2009-12-31.
[D1.4] "Runtime Environment", 2010-12-31.
[D2.6] "Implementation of an integrated prototype handling interactive user specified compliance requests in a compliance language", 2009-12-31.
[D4.2] "BPEL extensions for compliant services", 2009-12-31.
[D4.4] "Supporting infrastructure – process engine, process artefact repository, process generation tool", 2009-12-31
[D6.1] "Use Case, Metrics and Case Study", 2008-07-31

7.2. External documents

[Fow10] Martin Fowler, Domain-Specific Languages, Addison-Wesley Professional, 2010
[OZD10] E. Oberortner, U. Zdun, S. Dustdar, Patterns for Measuring Performance-Related QoS Properties in Distributed Systems. In Proceedings of the Pattern Languages of Programming Conference 2010 (PLoP 2010), Reno, Nevada, USA, ACM, October, 2010.

[3] http://ant.apache.org

[Zdu10] U. Zdun. A DSL Toolkit for Deferring Architectural Decisions in DSL-Based Software Design. Information and Software Technology. *Software & System Modeling*, vol. 52, no. 9, pages 733-748, Elsevier, 2010.