

## D4.4

Version: 2.0

Date: 2010-12-29

Dissemination status: PU

Document reference: D4.4



# Supporting infrastructure – process engine, process artefact repository, process generation tool

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 Technische Universitaet Wien (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 Università 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

**D4.4 Supporting infrastructure –  
process engine, process artefact repository, process generation tool**

Revision 2.0

Due date of deliverable: 2010-12-31

Actual submission date: 2010-12-29

Organisation name of lead partner for this deliverable:

USTUTT – Universitaet Stuttgart, Germany

Contributing partner(s):

TUV – Technische Universitaet Wien, Austria

UCBL – Université Claude Bernard Lyon 1, France

UNITN – Università degli Studi di Trento, Italy

THALES – Thales Services SAS, France

<b>Project funded by the European Commission within the Seventh Framework Programme</b>		
<b>Dissemination Level</b>		
<b>PU</b>	Public	X
<b>PP</b>	Restricted to other programme participants (including the Commission Services)	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission Services)	
<b>CO</b>	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	2009-09-06	All sections	New document based on D4.4 v1.0	USTUTT
0.2	2010-10-18	Section 1,2	Revised and updated content of former version of D4.4	USTUTT
0.3	2010-10-20	Section 4	Revised and updated content, added new screenshots, described new functionality	USTUTT
0.4	2010-10-28	Section 4, 5, 8	Revised and updated content, added new screenshots, described new functionality	TUV
0.5	2010-11-15	Section 6	Added content	UCBL
0.6	2010-11-18	Section 3	Revised and updated content of former version of D4.4	USTUTT
0.7	2010-11-25	All sections	Prepared for internal review	USTUTT
0.8	2010-12-06	All sections	Revised	USTUTT, UNITN
0.9	2010-12-13	All sections	Revised	USTUTT, THALES
0.91	2010-12-17	All sections	Prepared for approval	USTUTT
1.0	2010-12-24		Approval	TUV

## Authorisation

No.	Action	Company/Name	Date
1	Prepared	USTUTT	2010-12-17
2	Approved	TUV	2010-12-24
3	Released	TUV	2010-12-29

Disclaimer: The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies.

### All rights reserved.

The document is proprietary of the COMPAS consortium members. No copying or distributing, in any form or by any means, is allowed without the prior written agreement of the owner of the property rights.

This document reflects only the authors' view. The European Community is not liable for any use that may be made of the information contained herein.

## Contents

1. Introduction .....	7
1.1. Purpose and scope .....	7
1.2. Document overview .....	8
1.3. Definitions and glossary .....	8
1.4. Abbreviations and acronyms .....	8
2. Integration into the COMPAS architecture .....	9
3. Process engine prototype description .....	12
3.1. Requirements and specification .....	12
3.2. Traceability Information .....	13
3.3. Implementation.....	13
4. Process artefact repository prototype description .....	18
4.1. Process and process fragment repository Fragmento .....	18
4.1.1. Requirements and specification .....	18
4.1.2. Design and implementation.....	23
4.2. Model-Aware Repository and Service Environment .....	31
4.2.1. Motivation from the MDD perspective.....	32
4.2.2. Model-Aware Service Environment.....	33
4.2.3. Model repository .....	34
4.2.4. Model-aware services.....	36
4.3. Integration of Fragmento and Morse.....	38
5. Process generation tool prototype description .....	39
5.1. Requirements and specification .....	40
5.2. Design and implementation.....	40
6. Generation of Business Protocols .....	41
6.1. Communication activities.....	42
6.1.1. Receive activity .....	42
6.1.2. Reply activity .....	42
6.1.3. Invoke activity.....	43
6.2. Structured activities.....	43
6.2.1. If activity .....	43
6.2.2. Sequence activity.....	44
6.2.3. RepeatUntil activity.....	44
6.2.4. Flow activity.....	45
7. Roles of prototypes and usage in the COMPAS usage scenarios .....	45

7.1. The ICT security scenario .....	45
7.2. Advance telecom service for Mobile Virtual Network Operators .....	46
8. Prototype websites.....	46
9. Reference documents .....	46
9.1. Internal documents .....	46
9.2. External documents .....	47

### List of figures

Figure 1 Overall COMPAS architecture .....	11
Figure 2 Architecture of the Pluggable Framework.....	14
Figure 3 Flow of the traceability information to the compliance custom controller.....	15
Figure 4 Event generation during execution of a process .....	15
Figure 5 Conceptual model for Fragmento .....	20
Figure 6 Versions of an artefact .....	21
Figure 7 Model of the version management.....	22
Figure 8 Relations among artefacts .....	22
Figure 9 Fragmento package structure .....	24
Figure 10 Start page of the Fragmento web client.....	26
Figure 11 Artefacts management in Fragmento web client.....	27
Figure 12 Relations management in Fragmento web client .....	27
Figure 13 Display of an artefact in Fragmento web client .....	28
Figure 14 Wizard for creation of relations and annotations .....	29
Figure 15 Project structure of the test suite of Fragmento.....	30
Figure 16 Process view transformations in Fragmento .....	31
Figure 17 The MORSE Logo.....	31
Figure 18 Overview of the Model-Aware Service Environment.....	33
Figure 19 Architecture of the MORSE repository .....	33
Figure 20 MORSE objects and projects .....	34
Figure 21 MDD artefacts (left part a) and additional MORSE objects (right part b).....	35
Figure 22 Model element and model relations .....	35
Figure 23 Sequence diagram of a model-aware service .....	37
Figure 24 Overview of the process generation tool.....	40

### List of tables

Table 1 Overview of MORSE availability over Web services and RESTful services .....	38
---	----

Table 2	Model-aware service operations .....	38
---------	--------------------------------------	----

### **List of listings**

Listing 1	Example of a traceability element in XML.....	13
Listing 2	Example of an execution event emitted .....	17
Listing 3	Synchronization of new artefacts.....	39

## Abstract

This deliverable provides a revised description of the technical infrastructure to support the design, generation and execution of compliant processes. The infrastructure consists of a process engine, two process artefact repositories and a process generation tool.

To support traceability during execution we present an extension of the open-source process engine Apache ODE. To support reusable units for compliance, namely compliance fragments, we present the fragment-oriented repository Fragmento. To support compliance-driven models we developed the model-aware repository MORSE. To support monitoring of business protocols we present a way to generate a business protocol specification from a business process specified by means of BPEL. Furthermore, we discuss an integral part of the View-based Modeling Framework which generates process descriptions and service descriptions of processes.

This deliverable is an update of the former version of D4.4 from M23. In order to make this deliverable self-contained most of the prototype descriptions of the former version have been included in this document. All descriptions have been revised and updated with respect to changes of the prototypes and new features which have been implemented.

## 1. Introduction

Nowadays, compliance is an important challenge for companies, independent from the branch of business. In reaction to financial and other scandals, e.g., concerning balance sheet fraud, companies are forced to conform to laws and regulations. These are not exclusively demanded by laws such as Basel II [BCBS06] which is setting up rigorous risk and capital management requirements in order to protect the international financial system from problems that might arise in case a major bank or a series of banks collapse. Also regulations and internal policies contribute to the large body of requirements that companies have to adhere to, in order to be compliant.

The goal of compliance requirements, e.g., demanded by laws, is to enhance the transparency of business decisions and to expand the accountability of responsibilities. Since the actual business of a company (i.e., production, selling or delivery of products and services) is directly related to its business processes, compliance to regulations implies *compliance-aware* business process management (BPM). As stated in [SLM+10, FCD+09] there is a lack of automated tool support for augmenting business processes with compliance available that enables compliance-aware business process management in process design time and runtime.

To address this gap, we present in this deliverable the prototypes to enable compliance-aware BPM, namely the process generation tool, the process artefact repositories along with the process engine. All these prototypes belong to the supporting infrastructure for compliance-aware business process management, which is part of the COMPAS framework.

### 1.1. Purpose and scope

The purpose of this deliverable is to describe the final versions of four COMPAS prototypes. These are (i) the process engine, the process artefact repository which consists of (ii) the process (-fragment) repository Fragmento and (iii) the Model-Aware Repository & Service Environment MORSE, (iv) the generation of business protocol specifications, and (v) the part of the View-based Modelling Framework (VbMF) that enables the generation of processes.

The scope of this deliverable is the infrastructure that supports usage and management of reusable artefacts for augmenting processes with compliance and their respective execution. As proposed in a previous deliverable [D4.1], we are using process fragments for compliance for this purpose (the short form is compliance fragments). In addition, we show and explain the application of the supporting infrastructure regarding the COMPAS usage scenarios.

## 1.2. Document overview

This document specifies and describes the final version of the supporting infrastructure for usage of process fragments for generating compliant process models during design time, as well as their execution during runtime. Therefore, we begin with used terminology and abbreviations in the introduction section. Then, we give an overview how the different prototypes described within this deliverable are embedded and integrated in the overall COMPAS architecture in Section 2. The main sections specifying and describing the initial versions of the COMPAS prototypes are “Process engine prototype description” (Section 3), “Process artefact repository prototype description” (Section 4) and “Process generation tool prototype description” (Section 5). After description of the prototypes we show their role and the conceptual integration by applying them to both COMPAS usage scenarios.

## 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> section terminology. This helps to make the overall COMPAS approach more comprehensive for the general public.

In the following the definitions of terms valid only in the scope of this deliverable (and therefore not listed on the public COMPAS Web-Site) are specified. To offer a self-contained deliverable, general terms of the COMPAS terminology are copied here.

*Compliance fragment:* A Compliance fragment (i.e., a process fragment for compliance) is a connected process structure that can be used as a reusable building block to ensure a consistent specification of compliance regarding a business process. Compliance fragments can be used to implement a compliance rule in terms of activities and control structures.

*Compliance requirement:* A constraint or assertion that results from the interpretation of the compliance sources. It may be defined in various levels of abstraction.

*Compliance target:* A Compliance target is a generic specification, such as a business process, or a compliance fragment, which is a target of compliance requirements.

## 1.4. Abbreviations and acronyms

BPEL	Business Process Execution Language
CEP	Complex Event Processing
CLRT	Compliance Language Request Tools
DSL	Domain-Specific Language
MDD	Model-Driven Development

MDS	Model-Driven Software Development
MORSE	Model-Aware Repository & Service Environment
oAW	openArchitectureWare
ODE	Orchestration Director Engine
SQL	Structured Query Language
UUID	Universally Unique Identifier
VbMF	View-based Modelling Framework
WSDL	Web Services Description Language
XML	Extensible Markup Language

## 2. Integration into the COMPAS architecture

In this section, we describe how the five prototypes described in this deliverable are placed in the overall COMPAS architecture and how they integrate and connect other COMPAS components. These prototypes are an extended process engine, process artefact repositories, a process generation tool, and a tool for calculation of business protocols.

The process generation tool is component that is tightly integrated in the MDS Framework, because this is the main point of integration for all domain-specific languages (DSLs) developed within the COMPAS project. The MDS framework is an essential part of the COMPAS architecture, addressing design time and consists of three components, namely DSL Transformation component, View-based modelling framework and the Code Generator, cf. Figure 1 (left). The DSL Transformation is used for the creation of compliance requirements in the corresponding DSL developed in COMPAS, e.g., the licensing DSL, quality of service DSL or security DSL. The compliance requirements are stored in the Model Repository (described in detail in Section 4.2) or are directly transferred to the VbMF.

After receiving the model instance of compliance requirements (specified using DSLs) or retrieving them from the Model Repository, the VbMF is utilized to compose view models at two levels of abstraction. Abstract view models are used for representing business- and domain-oriented concepts and knowledge whilst the technology-specific view models are leveraged for aligning these concepts with the IT infrastructure. Both types of view models describe the corresponding aspects concerning business processes and services. During the description and creation of views, existing compliance fragments, related artefacts or whole processes might be retrieved from the Process (-fragment) Repository Fragmento. The annotation editor is an integral part of Fragmento. It is used to create relations between the compliance fragments and other artefacts stored in Fragmento, e.g., WS-Security policies. A detailed description of Fragmento can be found in Section 4.1.

The view model created with VbMF serves as input to the Code Generators which are based on the template-based code generation techniques provided by the Xpand language of openArchitectureWare (oAW) [Eclipse09]. Thus, the Code Generators take the view models as well as templates as input and create complete business processes in BPEL including the WSDL file (that contains the interface descriptions) and deployment descriptors. The generated processes including information required for deployment can afterwards either be stored in the Process (-fragment) Repository Fragmento or can be directly deployed on the

process engine Apache Orchestration Director Engine (Apache ODE) [Apache09a] for execution.

The Model Repository (which is part of MORSE) and the process (-fragment) repository Fragmento are integrated with each other by means of a synchronization mechanism. Thereby, a direct relation between the DSL model instances and the view models stored in the Model Repository and the BPEL processes and process fragments stored in Fragmento is realized by using unique identifiers. Therefore, it is possible to retrieve process fragments and processes stored in Fragmento by querying the Model Repository. This is required for the interaction and integration of the Model Repository with the Compliance Language Request Tools (CLRT), cf. Figure 1. The CLRT queries the Model Repository and retrieves compliance targets, i.e., a process, or a compliance fragment. Afterwards, the static compliance checking is performed by using the support and functionality provided by the components for process verification. The results of the compliance checking during design time are thereafter stored in the Model Repository.

As already stated, the processes include all required information for deployment and execution and can be directly deployed to the process engine by the Code Generator Components. For enabling traceability the concept of Universally Unique Identifiers (UUIDs) is utilized. With this concept we also build up the relation between the design time and runtime that is required for monitoring, e.g., for drill-down to root cause of a compliance violation. Each artefact stored within the Model Repository has a UUID, which is included in the BPEL file of the process. The process engine has been extended to ensure that the UUIDs required for traceability are contained in the events emitted by the process engine during execution of a process. Thus, this information is provided to the compliance governance part of the overall COMPAS architecture. There it is used for querying the Model Repository for detailed information in case a compliance violation has been detected.

The Business Protocol Monitoring requires an abstraction of the processes in order to specify the properties to be monitored. This abstraction is called business protocol. The purpose of the business protocol abstraction is essentially to specify the set of conversations, i.e., a sequence of messages that are supported by a business process. To enable this kind of monitoring we present a prototype and concept to generate a business protocol specification from a business process specified by means of BPEL (see Section 6).

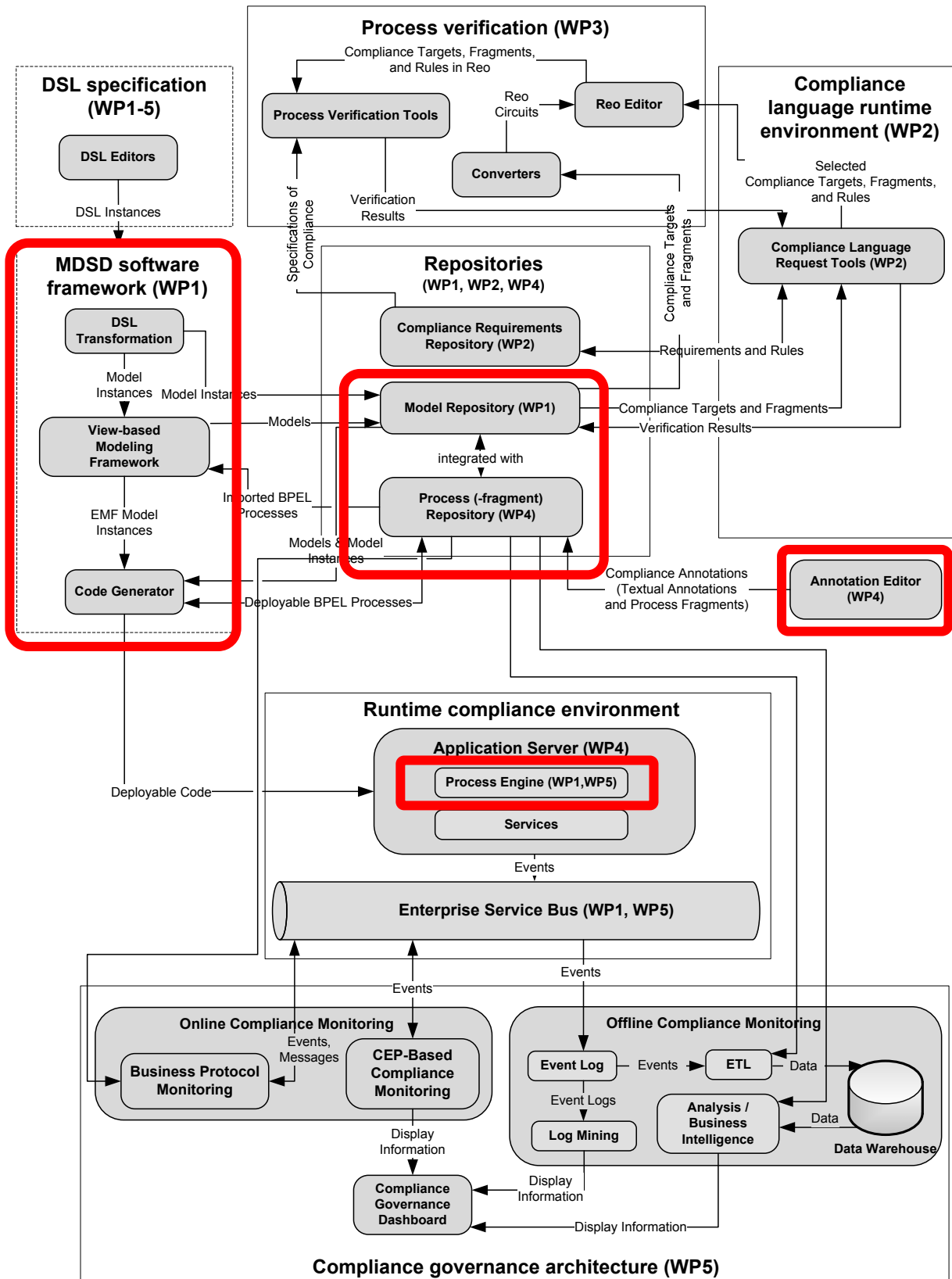


Figure 1 Overall COMPAS architecture

### 3. Process engine prototype description

In this section we describe the final process engine and respective engine, which is used for the execution of business processes represented in BPEL which have been augmented with compliance (cf. BPEL extensions for compliant services, [D4.2]). In comparison to the previous version, some parts of the implementation changed significantly. Instead of tightly coupling the extension with the process engine, we build on a framework enabling a more loosely coupled approach. The framework is called “pluggable framework for enabling the execution of extended BPEL behaviour” [KKL07]. It is portable to other engines. As other engines have another internal architecture and process navigation, the code of the pluggable framework has to be adapted for each engine. The events emitted, however, remain the same. Thus, our extension can be used without modifications at other process engines supporting the pluggable framework.

#### 3.1. Requirements and specification

According to the Description of Work for COMPAS [DoW] the main requirement is to develop or extend an existing process engine, which is distributed under an open source license. Thus, the possibility for use and adaption in industry and other research activities, both within the COMPAS project duration and beyond, is increased. Moreover, the access to the source code and also the extension of the code for COMPAS purposes is eased under open source licenses.

In addition to the basic requirements each known available BPEL process engine fulfils, we investigated the following additional requirements regarding the execution of BPEL processes augmented with compliance:

- Requirement 1: The process engine has to be capable of handling the **BPEL traceability** extension introduced in Section 5.2 of [D4.2]. This includes the integration of the required information for traceability, e.g., the UUID, into the events emitted during runtime.
- Requirement 2: For enabling the monitoring and mining of both COMPAS usage scenarios the events emitted by the engine during execution have to **provide all information required** to generate high-level events. In terms of implementation, this means that sufficient events are published and that these events carry sufficient information. We have investigated the information required for monitoring of all compliance requirements of the WatchMe scenario. They are listed in Table 4 within COMPAS deliverable D5.4.

Please note that not all compliance requirements are monitored based on the events published by the process engine, because in COMPAS we follow different approaches for dynamic checking of compliance during and after runtime. Elementary compliance requirements are checked online using Complex Event Processing (CEP) rules in combination with the complex event processing engine Esper [ESPER09]. Advanced compliance requirements are monitored utilizing compliance annotation of business processes in combination with the components responsible for business protocol monitoring. Furthermore, offline monitoring is essential for long-running processes. For details please refer to [D5.4]. All information relevant for monitoring and mining of compliance requirements during and after runtime are provided in this document.

## 3.2. Traceability Information

The embedding of traceability information in the BPEL process remained unchanged in comparison to [D4.4a]. Listing 1 shows an example of a traceability information element that is integrated in the business process.

```
<morse:traceability build="dc3d74f6-1670-4113-a8cb-13b1cb0f400f">
  <row query="/process"
    queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
    <uuid>bdf58925-a832-46d3-884f-b9ed85be3978</uuid>
    <uuid>45aa9322-cb8d-499e-b8ad-877c251c8fc0</uuid>
    <uuid>cc48bf23-2690-4013-a202-1e220c5a3525</uuid>
  </row>
  <row query="/process/sequence[1]/receive[2]">
    <uuid>376cbd64-5a4c-3c27-a919-4dacb9d42832</uuid>
  </row>
</morse:traceability>
```

**Listing 1 Example of a traceability element in XML**

Any `<morse:traceability>` element has a property `build`, which contains a UUID referencing generation information stored in the Model Repository of the corresponding generation of the BPEL process in which' definition (BPEL file) this `<morse:traceability>` element is integrated. The generation of BPEL processes augmented with compliance is performed by the VbMF and the Code Generators which are part of the MDSO software framework, cf. Figure 1 and see Section 5.

One `<morse:traceability>` element consists of one or more `<row>` elements, cf. Listing 1. Each particular `<row>` element represents the traceability information concerning one concrete element of the BPEL process. This element within the BPEL process is uniquely identified by the content of the properties `query` and `queryLanguage` of each `<row>` element. The value of the property `queryLanguage` defines the type of query language used for specifying the query for identifying the corresponding element within the BPEL process. In the example given in Listing 1 the XML Path Language (XPath) Version 1.0 is used. The property `query` contains the query specified in the given query language to identify the corresponding element.

In the first `<row>` element in Listing 1 the process element is identified by the query. Due to the fact, that more than one view model is used as input for the generation of a BPEL process by the Code Generators, there are three different UUIDs contained in the first `<row>` element, referencing three different view models stored in the Model Repository.

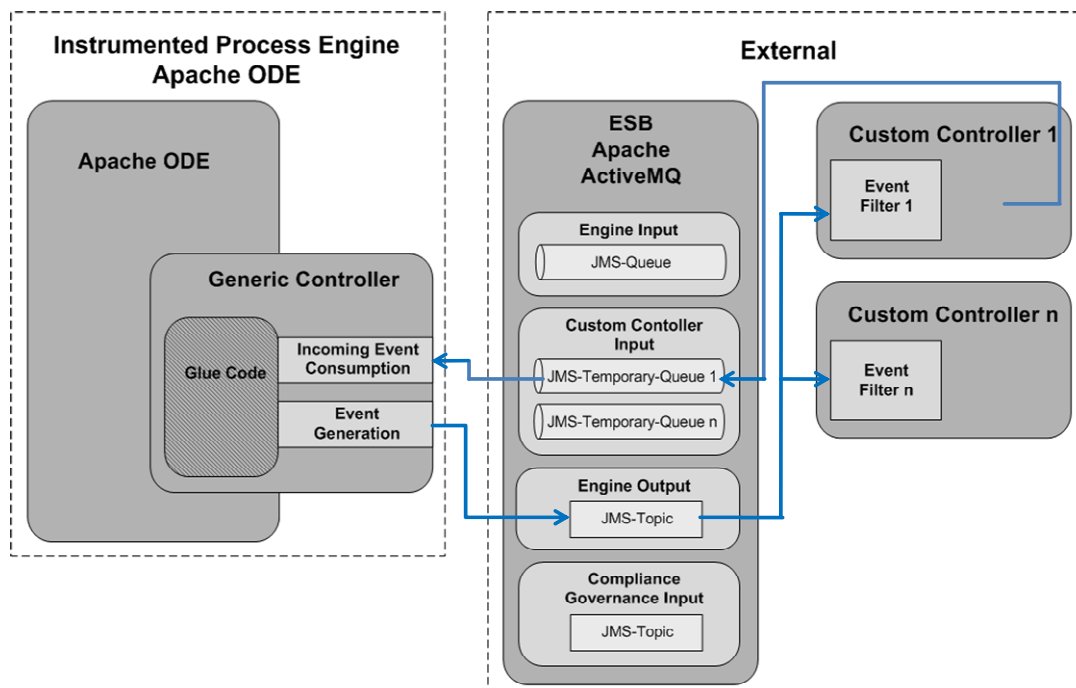
In the second `<row>` element a receive activity is referenced by the `query` property. The corresponding information stored in the Model Repository is referenced by a UUID.

## 3.3. Implementation

Considering the requirements above, we decided to use the Apache ODE [Apache09a] as basis and to extend it for our purposes in COMPAS. Apache ODE is open source and provided under the Apache License Version 2.0 [Apache04]. We integrated the “pluggable

framework for enabling the execution of extended BPEL behaviour” [KKL07] into the Apache ODE engine. The main idea of this pluggable framework is to provide generic BPEL events which are independent of the used engine. The events for BPEL 1.1 are listed in [KKS+06] and the ones for BPEL 2.0 in [Ste08]. This extension also enables modification of the process behaviour during runtime. As in COMPAS, we do not focus on compliance enforcement, we do not make use of the interfaces provided to react to violations by changing the behaviour of the BPEL process. We use the emitted events for passive monitoring only. An implementation is available for ActiveBPEL [KKL07] and Apache ODE 1.1.1 [Ste08]. Since the last version of D4.4, we ported this framework to Apache ODE 1.3.4 as the final scenario implementations rely on functionality provided by that version of ODE.

The architecture of the pluggable framework is presented in Figure 2. A generic controller is responsible for the events in the generic event model. The glue code ties between the generic event model and the native navigation model and capabilities of the engine. The event generation and event consumption capabilities enable the Generic Controller to communicate with pluggable Custom Controllers via these events. A Custom Controller does the work related to a specific extension: it filters the events produced by the Generic Controller to only those that are relevant for its specific application domain, and provides the necessary extension behaviour [KKL07].



**Figure 2 Architecture of the Pluggable Framework**

In the following we explain and specify how the prototype handles the traceability information integrated in the BPEL process by MORSE. We implemented a custom controller, called “compliance custom controller”, which consumes the ODE events, enriches them with traceability information and publishes them on a JMS topic in the enterprise service bus [D1.4]. This is illustrated in Figure 3 and Figure 4. Figure 3 illustrates how the custom controller gets aware of the traceability information. As soon as a process is deployed, a “process deployed” event is published on the JMS-Topic “Engine Output”. This event contains the BPEL process which in turn includes the traceability information. This event is consumed by the compliance custom controller. The compliance custom controller stores the mapping information from activity XPathS to the activity UUIDs. Each event of a process

instance contains a XPath to the activity where that event belongs to. As soon as an event is received, the mapping from XPath to the activity UUID is retrieved and is used for determining the UUID of the XPath of the event. Using that information, a new event is generated. That event is in XML format and contains all information of the received event plus the process UUIDs and the UUID belonging to the XPath.

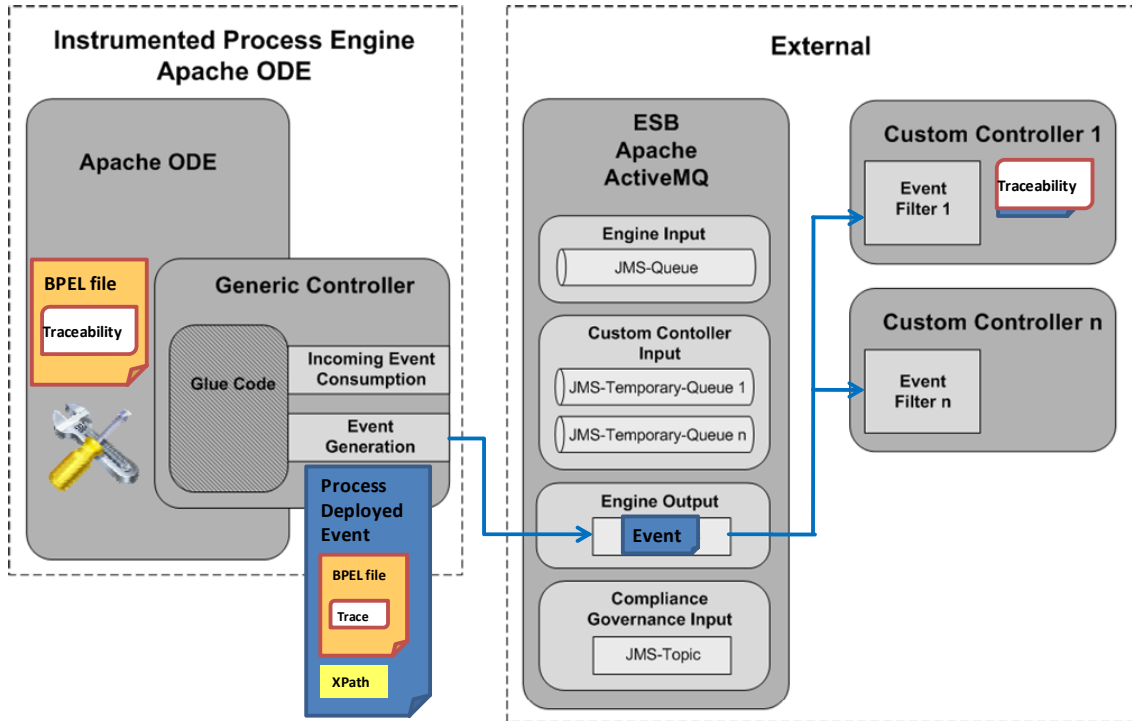


Figure 3 Flow of the traceability information to the compliance custom controller

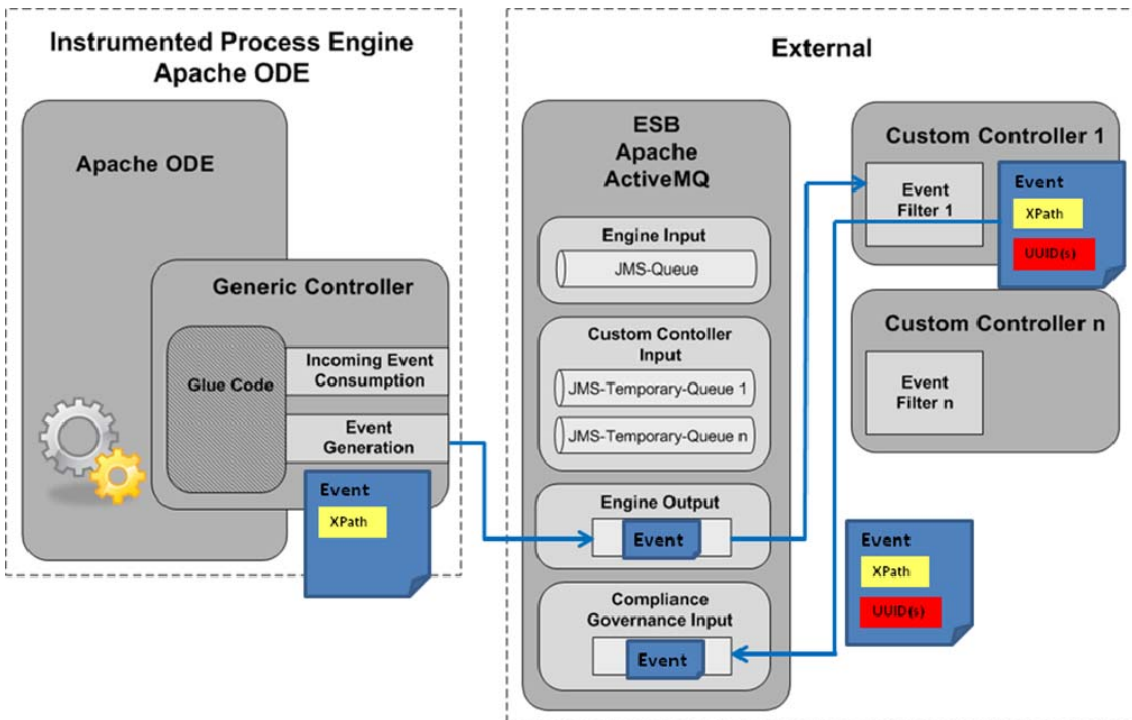


Figure 4 Event generation during execution of a process

Listing 2 shows an example of an execution event, which is emitted by the compliance custom controller.

```
<Event xmlns="http://xml.infosys.tuwien.ac.at/ns/compas/event">
  <timestamp>2010/11/22 23:02:33</timestamp>
  <source>Process Engine</source>
  <type>ActivityExecutingEvent</type>
  <name>ac0af35f-45a5-4907-ba02-a67d1abc573e</name>
  <processBuildUUID>268a3509-bbca-4640-a00a-9ab2ca2592c1
  </processBuildUUID>
  <processInstanceId>3251</processInstanceId>
  <property name="messageID">27</property>
  <property name="activityName">Check user data</property>
  <property xmlns:ns3="http://www.compas-ict.eu/watchme"
    name="processName">ns3:WatchMeProcess
  </property>
  <property name="processUUIDList">
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">5a3b5
7ac-ac45-4381-8f79-0853ab90e58a
  </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">f2be4
55d-aceb-4c8c-9eca-0c71d24f4ed0
  </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">445f2
8ff-0be5-4019-ae30-e03cccfbea13
  </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">594bb
ecc-5df2-4afb-9f13-4b3e54381f9f
  </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">13878
820-c310-493c-8d21-4409dfcdbc1c
  </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">5cd1d
8ea-d0ef-44c7-b6dc-16adbaec9418
  </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">c2c53
4c0-2af1-4afd-a14f-fb8d006d965a
  </ns2:uuid>
  </property>
  <property name="processVersion">5</property>
  <property name="activityInstanceID">3302</property>
  <property
name="activityXPath">/process/sequence[1]/pick[1]/onMessage[1]/sequence[1]/
invoke[1]
  </property>
  <property name="scopeInstanceID">3301</property>
  <property name="scopeXPath">/process</property>
  <property name="uuid">50b9f4d6-624c-35ba-a617-1736dee0dc17</property>
  <property xmlns:ns="http://www.compas-ict.eu/watchme"
name="portType">ns:UserDataCheck
  </property>
  <property name="operation">checkUserData</property>
</Event>
```

```

<property name="endpointReference">
  <service-ref xmlns="http://docs.oasis-open.org/wsbpel/2.0/serviceref">
    <EndpointReference xmlns="http://www.w3.org/2005/08/addressing">
      <Metadata>
        <ServiceName
xmlns="http://www.w3.org/2006/05/addressing/wsd1"
          xmlns:servicens="http://www.compas-ict.eu/watchme"
EndpointName="UserDataCheck">servicens:UserDataCheckService
        </ServiceName>
      </Metadata>
      <Address>
http://localhost:8080/UserDataCheckWS/services/UserDataCheck
      </Address>
    </EndpointReference>
  </service-ref>
</property>
</Event>

```

### Listing 2 Example of an execution event emitted

We ported the pluggable framework from ODE 1.1.1 to ODE 1.3.4. The main technical difficulty resulted from a different job scheduling. The implementation of the compliance custom controller consists of following packages:

- *eu.compas\_ict.compliance\_custom\_controller*,
- *eu.compas\_ict.compliance\_custom\_controller.messages.controllerOut*,
- *org.apache.ode.bpel.extensions.comm.messages.engineOut*.

In the following we summarize the content of these packages and provide information on the important classes.

- Package *org.apache.ode.bpel.extensions.comm.messages.engineOut*

This package contains one class for each event type that can be emitted by the ODE engine

- Package *eu.compas\_ict.compliance\_custom\_controller.messages.controllerOut*

This package contains one class for each event type that can be emitted by the compliance custom controller and which is sent to the governance input topic. These classes extend the classes of *comm.messages.engineOut* by adding fields containing traceability information.

- Package *eu.compas\_ict.compliance\_custom\_controller*
  - *MsgDispatcher.java* This class handles incoming messages. For each type of message, an object of *messages.controllerOut* is instantiated. The data of the incoming message is copied to the new object. The traceability information is fetched from the storage and added to the event. A hashmap is used to store the mapping from an element to the traceability information. After adding the traceability information, the event is sent to the governance input topic by the class *Communication*.

- `Communication.java`: This class transforms the message to be sent into an XML document following the COMPAS event format [D5.4] and sends it to the governance input topic.

## 4. Process artefact repository prototype description

For the process artefact repository we have made the architectural decision to split it up into two components. These components are (i) `Fragmento`, which is the process and process fragment repository, and the (ii) `MORSE`. The first one, “`Fragmento`”, is a repository that is dedicated to the management of process-related artefacts, such as BPEL processes, WSDL documents, deployment descriptors, and especially, compliance fragments. The second repository, “`MORSE`”, is a general purpose repository for storing almost any kind of model. However, it does not provide special functionality for the management reusable units of processes. `Fragmento` provides particular functionality in addition to the basic functionalities (such as version management) for handling compliance fragments and related process artefacts. This additional functionality also represents the rationale for splitting the storage of models in two different repositories.

We have identified several functions that are particularly helpful in the management of process artefacts. Firstly, process design and process enactment require valid models for proper execution, thus `Fragmento` provides XML schema validation, and provides an extensibility mechanism for integration of additional validation functions. Secondly, `Fragmento` supports view transformations for flexibly creating custom representations of processes and process fragments on the fly. Thirdly, `Fragmento` provides mechanisms for definition of bundles, which allows packaging of all artefacts related to a process (or fragment) together into one package. Fourthly, `Fragmento` provides a way for flexibly defining relations among artefacts, which is currently used for annotating a process fragment to a process. Furthermore, `Fragmento` provides an extensibility mechanism for custom query functions. This allows the implementation of search functions beyond the metadata of a process artefact (e.g., concerning the structure of a process fragment).

For making the artefacts which are managed in `Fragmento` visible and accessible in `MORSE` we have created an integration mechanism. We have defined a synchronization component that creates so-called proxy objects, which can be used like any other artefact stored in `MORSE`. This allows, for instance, linking a process fragment that implements a particular compliance requirement to its source, i.e. to the model of the requirement.

### 4.1. Process and process fragment repository `Fragmento`

#### 4.1.1. Requirements and specification

In this section we present “`Fragmento`”, the **Fragment-oriented** process artefact repository. `Fragmento` is dedicated to management of processes and process fragments for usage in the field of compliance. We begin by describing the basic requirements for the repository. Next, we sketch the code base that the repository is built on. Following this, we describe advanced functions and extensions that the repository supports. The application architecture and the concepts which are realized in `Fragmento` have been presented in the community [SKL+10]. Following discussions confirmed that the concepts and the supporting infrastructure we developed are a useful contribution to service-oriented information systems.

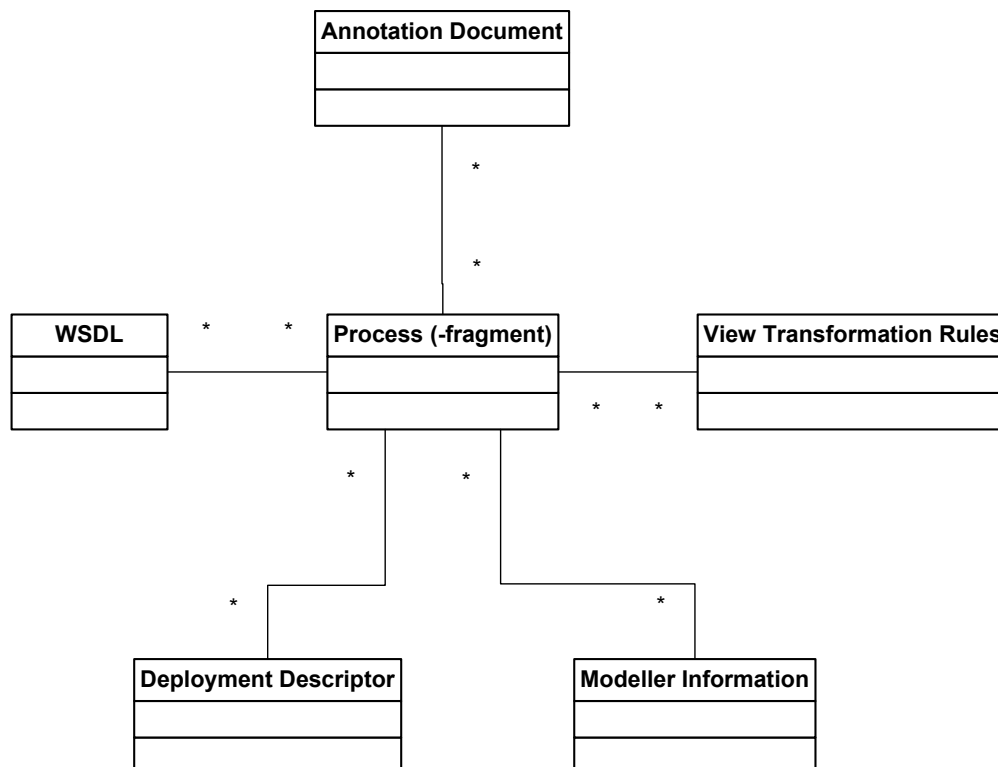
#### 4.1.1.i. Basic requirements

A repository for process artefacts should account for storage & retrieval and version management of all artefacts related to a process. In principle, it should support the CRUD operations: Create, Read, Update, and Delete. However, deleting contents from a repository may lead to broken references.

First of all, we can distinguish different types of artefacts that are related to a process or to a process fragment running in a service-based environment.

- A process or process fragment model, either in standard BPEL or in an extended or modified version of BPEL (cf. [D4.2], BPEL extensions for process fragments)
- An according WSDL document (for specifying the interface)
- A deployment descriptor according to the process or the process fragment
- Additional information for a process modelling tool (e.g., for storing graphical information for a modelling tool like (X/Y) coordinates of activities within a <flow> construct)
- A view transformation rule (explained in detail in Section 4.1.1.iii)
- Annotation documents (e.g., a policy specified in WS-SecurityPolicy )

The repository assigns unique identifier to each artefact being stored. The usage of unique identifiers allows creating relations between artefacts. The repository should provide an advanced management of relations and should furthermore allow annotation of process fragments to a given process for describing constraints, as we discussed in [D4.1]. The above listed types of artefacts are typically serializable to XML, therefore the repository should be able to store XML artefacts. Additional metadata, such as a description, keywords and a name are also applicable to all of the listed types. Figure 5 shows the conceptual model of the different types of artefacts and their relation.



### Figure 5 Conceptual model for Fragmento

From this analysis we can derive the requirements concerning the *model of the artefacts* that should be stored and managed in the repository. An artefact consists of the following parts:

1. A unique identifier
2. Metadata (Name, description, keywords etc.)
3. An XML document
4. A type (Fragment, WSDL etc.)
5. Relation(s) to other artefacts

For providing a flexible way to specify *relations* among artefacts a relation has to consist of the following parts:

1. A source (i.e. an artefact that is source of the relation)
2. A target (i.e. a different artefact)
3. A relation type (e.g. annotation)
4. A description for further characterization

Furthermore, efficient *searching* for artefacts is a fundamental requirement and should be well supported. Concerning search functionality we specify the following requirements:

- Search for a match in the description
- Search for a match in the content (i.e. within the XML document)
- Search by date of creation (by specifying an interval)
- Search by artefact type
- Search by date of creation, restricted to a particular type
- Search for artefacts which are related to a particular artefact

The repository should provide basic operations for the *management* of the artefacts:

1. Creation of an artefact
2. Checking out of an artefact (and locking the current version)
3. Releasing a lock (i.e. undoing the check out)
4. Checking in (and thereby creating a new version and possibly new relations)
5. Retrieving the version history of an artefact
6. Retrieving a particular version
7. Retrieving the latest version

For the management of the relation which can be defined among the artefacts, the repository should also provide basic operations:

1. Creation of a relation
2. Updating a relations
3. Deleting a relation

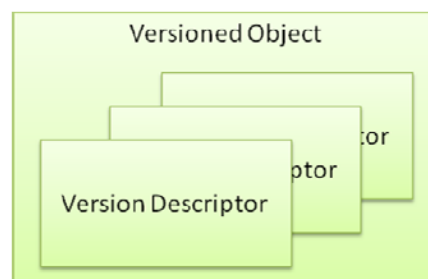
The definition of relations allows on the one hand specifying, which artefacts belong together. On the other hand they also allow defining an annotation of one artefact to another, e.g. the annotation of a process fragment to a process. According operations are required for resolving such relations. For instance, an operation for retrieving a complete process *bundle* would provide a comfortable way to retrieve a process and all related artefacts.

For integration with other components the repository should provide its *interfaces* as a Web Service, using WS-I basic profile for interoperability (see <http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html>), i.e. it should provide a SOAP/HTTP binding for the operations that it exposes. For easy integration with other tools, the operations for retrieving an artefact should be provided in a RESTful manner, i.e. accessible via HTTP/GET.

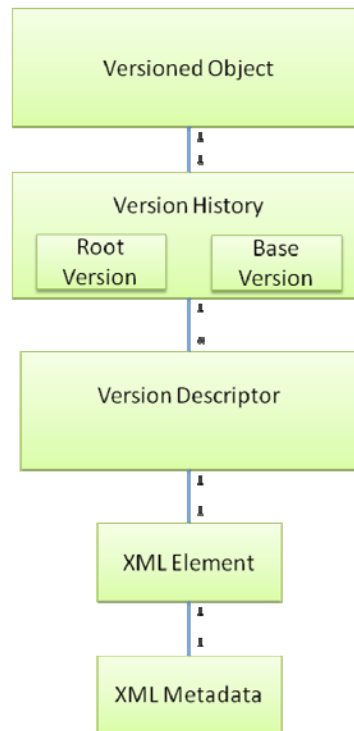
#### 4.1.1.ii. XML-oriented Repository as code base

Fragmento uses a code base for a repository that has been developed by the MASTER project (cf. [MASTER09]). The repository by MASTER is a general, though XML-oriented repository, and therefore well-suitable for the described purpose. The code base of this repository is open source (see [XML09], the project-specific extensions are not available but are also not required for running it).

When a new artefact is created, the repository creates a new “Versioned Object”. This object is the container for the different versions of the artefact (cf. Figure 6). The MASTER repository supports the model shown in Figure 7 for the versioning of artefacts. The Versioned Object does also provide a Version History object that allows accessing the root version (i.e. the first version) and the base version (i.e. the latest version). The internal representation of a version of an artefact is a “Version Descriptor” object. This is a container that stores the date of creation, metadata and a reference to the XML document of the artefact.

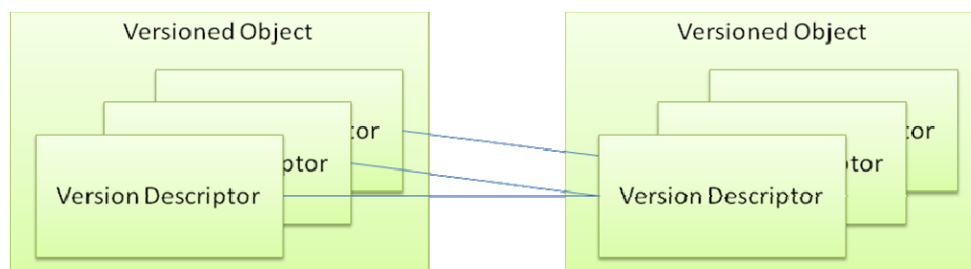


**Figure 6 Versions of an artefact**



**Figure 7 Model of the version management**

The MASTER repository does also allow creating relations among version descriptors (cf. Figure 8). This feature can be used for creating a bundle of artefacts, and it can also be used for creating an annotation of a fragment to a process, and for creating a textual annotation, respectively. It is important to note that the relations are created among Version Descriptors and not among Versioned Objects. This circumstance allows distinguishing between the relations of artefacts in different versions, but it also requires management of the affected relations when a new version of an artefact is created.



**Figure 8 Relations among artefacts**

#### 4.1.1.iii. Extensibility

Fragmento provides support for management of process artefacts which goes beyond the functionality of a general purpose repository. The additional functionality also represents the rationale for separating the storage of models in two different repositories. So far we have identified three different groups of functions that are helpful in the management of processes, process fragments and related artefacts:

1. Validators: The artefacts related to process enactment have to comply with particular requirements: a WSDL document has to conform to a given format and a BPEL

process has to comply with particular rules. The integration of custom validators provides a guarantee that an artefact complies with particular syntactical requirements. The validation can be executed as a prerequisite for check in, or otherwise be executed on demand.

2. **View Transformations:** In a view transformation particular transformations are applied to a process model. The view transformation implementation in Fragmento supports omission of activities or attributes. The outcome of a view transformation is referred to as “process view” [SLS10]. We have identified several scenarios, in which a view transformation supports the management of a process: A view on a process can be used for abstraction. This can for instance be used for providing a perspective on a process that is personalized for specific aspects that are relevant to an auditor, or for generating a public view on a process that hides confidential details.
3. **Custom query functions:** Beyond the metadata of an artefact (supported in Fragmento), other information is of interest for search, like the interfaces a process should provide, or the structure of a process fragment (not supported in Fragmento). The structure of a process fragment refers to the way in which process activities are connected together, arranged or organized [ML09]. Query of structural information is especially useful if the compliance encoded in the process fragments imposes constraints on the control flow of the process activities [SLM+10]. To support advanced query mechanisms, Fragmento provides an extension mechanism so that such functionality can be added.

## 4.1.2. Design and implementation

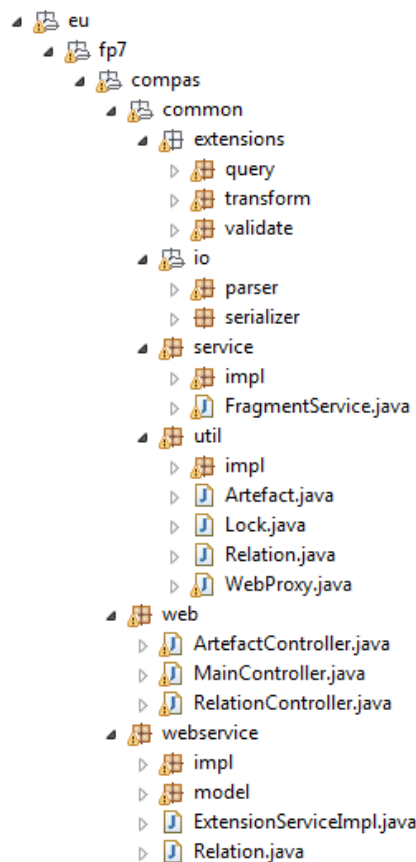
### 4.1.2.i. Application design

Concerning the backend, Fragmento is built on the technology stack that has been introduced by the MASTER repository [XML09]. That is, a Tomcat application server [Apache09c] which is hosting the repository application. Hibernate [Hibernate09] is used as data abstraction layer. Furthermore the Spring Framework [Spring09] is employed for object lifecycle management and a PostgreSQL database [Postgre09] is utilised for storage.

Concerning the development of the Web service interfaces, Axis 2 libraries [Apache09b] are used. The Web client is built using Java Server Pages (JSP) and Tag Libraries [Display09] for the view, while Servlets as used as the controller for handling client requests. Fragmento is overall written in Java.

Besides some minor code adaptations concerning the management of relations and the implementation of search functionality, the code base of the MASTER repository has been used and extended. Figure 9 shows an outline of the functionality that has been implemented for Fragmento in addition to the used code base. The `extensions` package hosts the extensions for custom queries, view transformations and custom validators. The `io` package hosts the XML parsers and serializers that are used for parsing and serializing XML messages that are sent to and from the Fragmento Web service. The `service` package holds the actual implementation of the Fragmento Web service, for details see Section 4.1.2.ii. The `util` package provides classes and functions that are utilized within the overall implementation. For instance, the `Artefact.java` class is utilized for display of an artefact on a web page. The `web` package contains the Servlets that manage client request of the web client. The `webservice` package contains the classes for the Web service interface which are automatically generated from the WSDL document of the Fragmento Web service. Most of

the additional code is concerned with the Web service interfaces and the Web client. The latter has been developed from scratch.



**Figure 9** Fragmento package structure

#### 4.1.2.ii. Web service interfaces

The Web service interface of the Fragmento repository provides the following operations:

Concerning artefacts:

1. `createArtefact`: This operation is used to create a new artefact in the repository. The operation returns the identifier of the created version descriptor.
2. `retrieveArtefact`: This operation is used to retrieve a particular version of an artefact, without performing a check out. This operation optionally allows specifying or referencing a view transformation rule that is applied to the artefact before it is returned.
3. `retrieveArtefactBundle`: This operation returns an artefact and all the artefacts that are related to it.
4. `retrieveArtefactHistory`: This operation returns a list of version descriptor identifiers that represent the version history of an artefact.
5. `checkoutArtefact`: This operation sets a lock on the requested artefact and returns it. It also returns a lock identifier which is required for check in.
6. `checkinArtefact`: This operation creates a new version of an artefact. For authorization also the corresponding lock identifier has to be passed. Based on the

parameter `keepRelations`, the relations of this artefact also apply to its new version (i.e. Fragmento creates new relations). It returns the identifier of the new version.

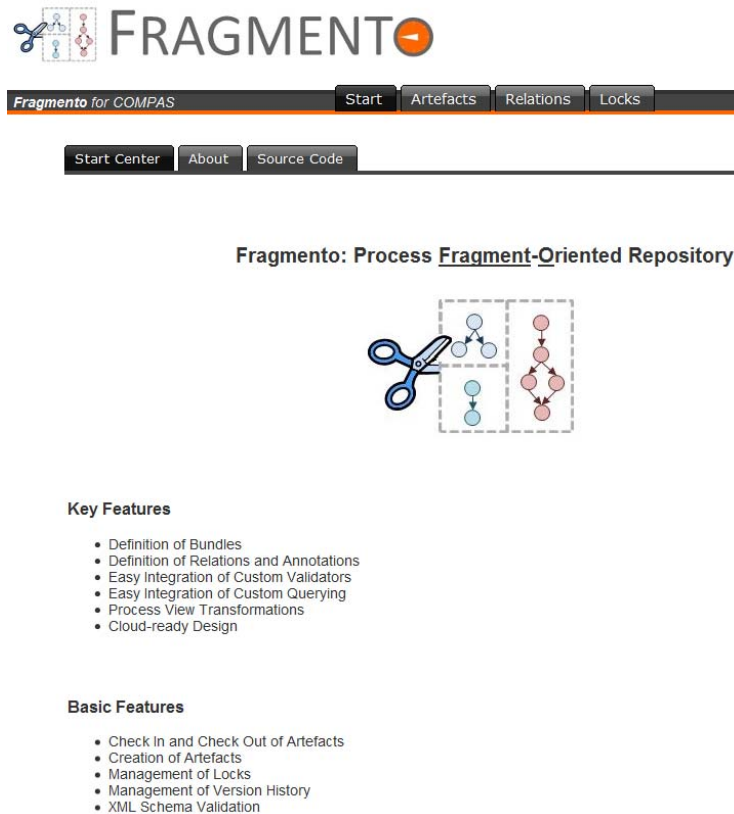
7. `browseArtefacts`: This operation implements the search function. Based on the input parameters this operation returns a list of version descriptors that match the query (considering only the head revision, i.e. the latest versions). As input parameters are accepted: artefact type, interval of creation, or search string for description or document content.
8. `retrieveArtefactLatestVersion`: This operation returns the latest version of an artefact. In order to retrieve past revisions, the operation `retrieveArtefact` has to be used.
9. `browseLocks`: This operation returns a list of all locked artefacts.
10. `releaseLocks`: This operation can be used to release a lock.

Concerning relations:

1. `createRelation`: This operation allows creating a relation from one artefact to another. The description can be characterized by a type and a description. This operation can also be used to create annotations.
2. `retrieveRelation`: This operation returns the details of a relation.
3. `browseRelations`: This operation provides search functionality for relations. Valid input parameters are a source (a version descriptor identifier), a target, a type, or an interval for its creation.
4. `updateRelation`: This operation provides the update mechanism for relations. As relations are not versioned, there is no check out or check in function.
5. `deleteRelation`: This operation completely deletes a relation.

#### **4.1.2.iii. Web client**

Fragmento provides all required functionality via Web service interfaces which can be exploited for integration with rich client applications, e.g., based on Eclipse. For easily accessing the Fragmento repository and managing the artefacts and relations stored therein we also provide a Web client. Figure 10 shows the start page of the Fragmento Web Client.



**Figure 10 Start page of the Fragmento web client**

We have created a double-stage navigation. On top level, the user can choose between the management of artefacts, relations or locks. On the second level the particular management functions for the according top level selection are shown. Figure 11 shows the user view for management of artefacts, and Figure 12 shows the management of relations. In the final version, the relations Annotation, Container, WSDL, Deployment, Modeller Data and Transformation are supported. These types implement the conceptual model of Fragmento.



Fragmento for COMPAS    Start    Artefacts    Relations    Locks

Manage    Search    Create    Check Out    Check In    Transform

Type	Artefact ID	Description	Show	Show Bundle	Show History	Check Out
WSDL	5691	trusted-timestamp-in-standard-code.wsdl	Show	Show Bundle	Show History	Check Out
WSDL	5684	thales.loanapprovalClient.001Artefacts.wsdl	Show	Show Bundle	Show History	Check Out
WSDL	5677	thales.loanapproval.001Artefacts.wsdl	Show	Show Bundle	Show History	Check Out
WSDL	5670	Approval-in-standard-code.wsdl	Show	Show Bundle	Show History	Check Out
WSDL	5663	Segregation-of-duty-in-standard-code.wsdl	Show	Show Bundle	Show History	Check Out
Transformation Rule	5656	Removal of Identifier extensions	Show	Show Bundle	Show History	Check Out
Process	5649	Loan Approval Client Draft	Show	Show Bundle	Show History	Check Out
Process	5642	Loan Approval Process Draft	Show	Show Bundle	Show History	Check Out
Modeller Data	5635	trusted-timestamp-in-standard-code.bpelx	Show	Show Bundle	Show History	Check Out
Modeller Data	5628	thales.loanapprovalClient.001.bpelx	Show	Show Bundle	Show History	Check Out
Modeller Data	5621	thales.loanapproval.001.bpelx	Show	Show Bundle	Show History	Check Out
Modeller Data	5614	Segregation-of-duty-in-standard-code.bpelx	Show	Show Bundle	Show History	Check Out
Fragment	5607	trusted timestamp	Show	Show Bundle	Show History	Check Out
Fragment	5600	trusted timestamp in standard BPEL	Show	Show Bundle	Show History	Check Out
Fragment	5593	segregation of duty; separation of duty; 4-eyes principle; in standard...	Show	Show Bundle	Show History	Check Out
Fragment	5586	segregation of duty; separation of duty; 4-eyes principle; BPEL4People	Show	Show Bundle	Show History	Check Out
Fragment	5579	Avoidance of infinite waits in standard BPEL	Show	Show Bundle	Show History	Check Out
Fragment	5572	approval fragment with extension identifiers	Show	Show Bundle	Show History	Check Out
Fragment	5565	approval fragment in standard BPEL code with extension identifiers	Show	Show Bundle	Show History	Check Out
Annotation	5558	WS-SecurityPolicy: User Name with Certificates, Sign, Encrypt	Show	Show Bundle	Show History	Check Out
Annotation	5551	WS-SecurityPolicy: Use of SSL Transport Binding	Show	Show Bundle	Show History	Check Out
Annotation	5544	WS-SecurityPolicy: UsernameToken without password	Show	Show Bundle	Show History	Check Out

Figure 11 Artefacts management in Fragmento web client



Fragmento for COMPAS    Start    Artefacts    Relations    Locks

Manage    Search    Create    Update

Relation Type	Relation ID	Relation Description	Relation From ID	Relation To ID	Update	Delete
modeller	5304	Modeller information for BPEL Designer	5222	5201	Update Relation	Delete Relation
modeller	5305	Modeller information for BPEL Designer	5229	5250	Update Relation	Delete Relation
modeller	5306	Modeller information for BPEL Designer	5236	5257	Update Relation	Delete Relation
modeller	5307	Modeller information for BPEL Designer	5201	5208	Update Relation	Delete Relation
annotation	5308	Security Policy	5138	5250	Update Relation	Delete Relation
annotation	5309	Security Policy	5138	5257	Update Relation	Delete Relation
wsdl	5310	WSDL for the fragment	5271	5201	Update Relation	Delete Relation
wsdl	5311	WSDL for the fragment	5278	5173	Update Relation	Delete Relation
wsdl	5312	WSDL for the process	5285	5250	Update Relation	Delete Relation
wsdl	5313	WSDL for the process	5292	5257	Update Relation	Delete Relation
wsdl	5314	WSDL for the fragment	5299	5208	Update Relation	Delete Relation
container	5347	A process	5250	5342	Update Relation	Delete Relation
container	5348	another process	5257	5342	Update Relation	Delete Relation
modeller	5517	Modeller information for BPEL Designer	5435	5414	Update Relation	Delete Relation
modeller	5518	Modeller information for BPEL Designer	5442	5463	Update Relation	Delete Relation
modeller	5519	Modeller information for BPEL Designer	5449	5470	Update Relation	Delete Relation
modeller	5520	Modeller information for BPEL Designer	5414	5421	Update Relation	Delete Relation
annotation	5521	Security Policy	5351	5463	Update Relation	Delete Relation
annotation	5522	Security Policy	5351	5470	Update Relation	Delete Relation
wsdl	5523	WSDL for the fragment	5484	5414	Update Relation	Delete Relation
wsdl	5524	WSDL for the fragment	5491	5386	Update Relation	Delete Relation
wsdl	5525	WSDL for the process	5496	5463	Update Relation	Delete Relation
wsdl	5526	WSDL for the process	5505	5470	Update Relation	Delete Relation

Figure 12 Relations management in Fragmento web client

We have also integrated a support for viewing the XML part of the artefact within the Web page, as shown in the artefact detail view in Figure 13.



Fragmento for COMPAS

Start

Artefacts

Relations

Locks

Manage

Search

Create

Check Out

Check In

Transform

## Show Artefact

Artefact Uid 5649  
 Artefact Type Process  
 Artefact Description Loan Approval Client Draft

Check Out

Open XML

## Artefact Content

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable" xmlns:tns="http://iaas.uni-stuttgart.de/compas/thales/001"
name="thales_loanapprovalClient.001" suppressJoinFailure="yes" targetNamespace="http://iaas.uni-stuttgart.de/compas/thales/001">
  <!-- Import the client WSDL -->
  <bpel:import importType="http://schemas.xmlsoap.org/wsdl/" location="thales_loanapprovalClient.001Artefacts.wsdl" namespace="http://iaas.uni-stuttgart.de/compas/thales/001"/>
  <!-- ===== -->
  <!-- PARTNERLINKS -->
  <!-- List of services participating in this BPEL process -->
  <!-- ===== -->
  <bpel:partnerLinks>
    <!--
    The 'client' role represents the requester of this service. It is
    used for callback. The location and correlation information associated
    with the client role are automatically set using WS-Addressing.
    -->
    <bpel:partnerLink myRole="thales_loanapprovalClient.001Provider" name="Customer" partnerLinkType="tns:thales_loanapprovalClient.001"
partnerRole="thales_loanapprovalClient.001Requester"/>
    <bpel:partnerLink name="Loan Approval Service"/>
  </bpel:partnerLinks>
  <!-- ===== -->
  <!-- VARIABLES -->
  <!-- List of messages and XML documents used within this BPEL process -->
  <!-- ===== -->
  <bpel:variables>
    <!-- Reference to the message passed as input during initiation -->
    <bpel:variable messageType="tns:thales_loanapprovalClient.001RequestMessage" name="input"/>
    <!-- Reference to the message that will be sent back to the
    requester during callback
    -->
  </bpel:variables>
</bpel:process>
```

**Figure 13 Display of an artefact in Fragmento web client**

As shown in Figure 14, we implemented a generic wizard, which guides the user through the different steps for creation of any type of the pre-defined relations: Annotation, Container, WSDL, Deployment, Modeller Data and Transformation. This wizard implements the annotation editor. There are basically five steps which have to be performed to create a relation. At first, the type of relation has to be chosen. In the next steps the source and the target of the relation have to be chosen. Then, additional descriptions can be entered. Having this information, the wizard can establish the desired relation between the chosen artefacts.

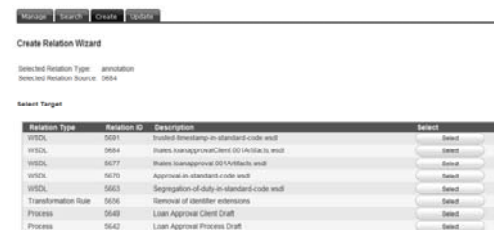
Step 1:  
Selection of the type of relation



Step 2:  
Selection of the source



Step 3:  
Selection of the target



Step 4:  
Description of the relation



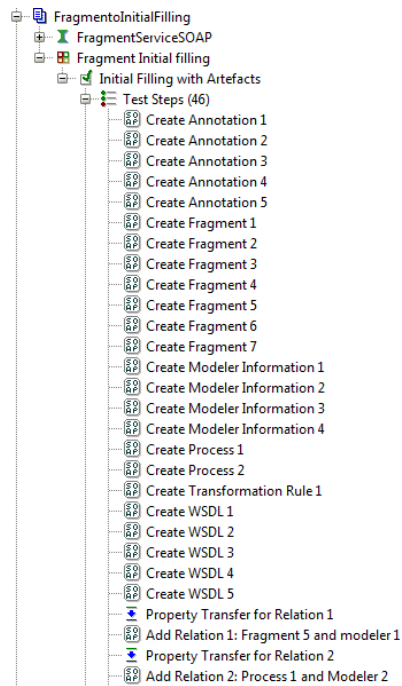
Step 5:  
Relation has been established



Figure 14 Wizard for creation of relations and annotations

4.1.2.iv. Initial filling of the Fragmento with artefacts

We created a test suite the can be used with SoapUI [SoapUI] to provide an initial filling of Fragmento with sample artefacts. The test suite (FragmentoInitialFilling-soapui-project.xml) can be easily imported in the tool via “file\import project”. Figure 15 shows the project structure of the test suite. First, annotation documents, fragments, processes and related artefacts are created in Fragmento. Then, relations between them are created by the test suite (annotation relations, bundle relations, etc.).



**Figure 15 Project structure of the test suite of Fragmento**

A simple right-click on the test suite “Fragment Initial filling” and selection of the test runner start the filling (“launch Testrunner \launch”). Then, the sample files are sent via SOAP/HTTP to the web service interface of the repository. The most recent version of the test suite is available at [Frgmto10]. This suite contains the latest versions of process fragments.

#### 4.1.2.v. Extensions

In Section 4.1.1.iii three groups of extensions were introduced, (i) custom validators, (ii) view transformations and (iii) custom query functions. In the final state of the prototype the following extensions are implemented:

Concerning custom validators, we have integrated a validator for checking XML against an XML Schema. Hence, we can use this kind of validation for checking a fragment against the XML Schema for BPEL Process Fragments (which we describe in deliverable [D4.2]). The validator has been configured for validation of the common artefact types used in Fragmento, e.g., WSDL documents.

Concerning view transformation we have integrated a process view transformation framework [SLS10]. We provide full support for one sample application: The application is to remove identifiers which have been added for traceability (cf. [D4.2], BPEL extension for traceability). This function transforms a process that was extended for traceability into standard BPEL code, in order to be usable in a standard process design or execution environment that does not provide support for this kind of extension. Figure 16 shows the Web client interface to access this function.

Rule ID	Description	Choose Transformation Rule	Show
5264	Removal of identifier extensions	Transform	Show
5477	Removal of identifier extensions	Transform	Show
5656	Removal of identifier extensions	Transform	Show

**Figure 16 Process view transformations in Fragmento**

The final version of the prototype does not contain an extension for custom queries. However, we provided the theoretical foundations and a discussion of implementation considerations [Pos10].

## 4.2. Model-Aware Repository and Service Environment



**Figure 17 The MORSE Logo**

We facilitate services to dynamically reflect on models with model-aware services (and components) and support these with a model repository. We call such a SOA a Model-Aware Service Environment (MORSE) [HZD09]. During the Model-Driven Development (MDD) process, each model of the SOA is placed in the model repository. Each model and model element gets a Universal Unique Identifier (UUID<sup>1</sup>) assigned, with which the model or model element can be uniquely identified. The UUIDs are generated into the source code of the model-aware services (and components). Hence, they are model-aware in the sense that they can retrieve the models from which they have been generated from the repository at runtime using a service. In the same way, other components such as monitoring, auditing, reporting, and business intelligence components, or MDD tools such as a model-driven generator, can retrieve these models.

The repository service interface is a generic interface, and the UUIDs are generically added to generated code. Hence, no changes of the generated SOA are necessary in order to use a model element at runtime that has not been used before. Also the model-aware services can access evolved models and model relationships that occurred after generation time of the model-aware service.

In the following sections we introduce MORSE, present the design of the repository, and introduce some model-aware services. We will describe how model-aware services interact

---

<sup>1</sup> Universal Unique Identifier (UUID) [ITU04] is a standard for unique identifiers in (distributed) software system development. UUIDs are used in MORSE to uniquely identify models and model elements across distributed components, such as the model-aware services, the model repository, and other components using the MORSE services such as monitors.

with the repository, how they can be created, and what functionality they may expose to other services.

### **4.2.1. Motivation from the MDD perspective**

In the broader view of model-driven systems in general, MORSE addresses two common problems in MDD systems: traceability and collaboration. In this section, we want to motivate both and explain briefly how the MORSE approach helps to address them. We describe these two common problems also to set the scope for this deliverable and delineate which parts of the general two problems are addressed by the model-aware services approach and which are not.

#### **4.2.1.i. Traceability in model-driven systems**

A particular problem of model-driven systems is traceability – asking the question: How do models and model elements of different abstraction layers and/or code correspond to each other? In particular, the traceability information for models that are transformed into other models or code can get lost in model-driven approaches. On the one hand, a transformation rule describes how source models are mapped to target models. On the other hand a traceability link at a target model would allow for identifying the source models. Traceability is essential for meaningful feedback from the runtime to stakeholders and for identifying and understanding the root cause, e.g., in case of a failure or exception. This is because, if we are able to trace the source model from which a target (model or code) has been created (via generation or transformation), it is possible to use the information in the source model to analyze or debug the target. MORSE helps to address this problem of traceability as it manages MDD projects and artefacts and relates them to UUIDs that are generated into target systems. Moreover, MORSE facilitates such systems to exploit their traceability links via UUIDs by querying and reflecting on models, model elements, and model relationships.

#### **4.2.1.ii. Support for collaboration in model-driven systems**

Most current tool support for model-driven development only focuses on the design time and comes with limited collaboration features, if any. Model-aware services, however, rather assume a distributed environment, maybe even distributed development. In order to facilitate various services of a distributed environment, however, to concurrently work with MDD projects and artefacts we need to support the management of projects and artefacts, with (1) versioning capabilities while capturing and keeping track of model relationships and (2) services for the information retrieval and the management of these. For (3) facilitating collaboration scenarios, we also need to (4) deal with concurrency, e.g., provide locking mechanisms, raise the awareness of the work of others, offer compare and merge possibilities as well as support for resolving conflicts. MORSE, as it is presented in this section, addresses the first two of these issues, versioning capabilities and services for information retrieval and management, as these features are also needed for the monitoring, auditing, reporting, and business intelligence purposes addressed by the model-aware services approach proposed in this document. For example, a monitoring component requires a service for retrieving model information from the MORSE repository, and it requires the models in the version of the model instance that it monitors. Please note that the collaboration features of MORSE could also be used for other scenarios, such as supporting the MDD process for distributed MDD.

### 4.2.2. Model-Aware Service Environment

For facilitating services to dynamically work with models in a SOA, we propose MORSE, the Model-Aware Service Environment. MORSE consists of a model repository and model-aware services that interact with the model repository using generic service-oriented interfaces. Figure 18 gives an overview of the MORSE. From the repository model-aware services can be generated that interact with the information retrieval interface. Also services with traceability information that emit events to model-aware services can be generated. In the following sections we will discuss these services in more detail.

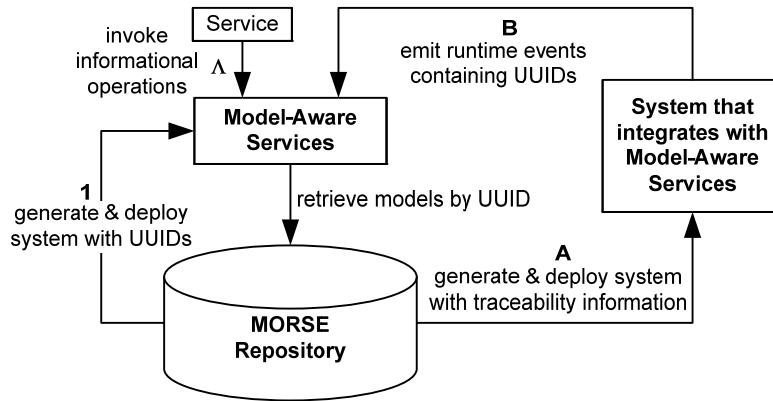


Figure 18 Overview of the Model-Aware Service Environment

Figure 19 gives a high-level overview of the model repository architecture. Different Web service interfaces allow for the administration and resource management of MDD projects and artefacts and offer information retrieval functionality (see end of the following section) to model-aware services. The MORSE builder service can create these model-aware services. Also it can weave UUIDs of MORSE objects into generated code. A deployment service is used for deploying resulting services and processes on runtime engines.

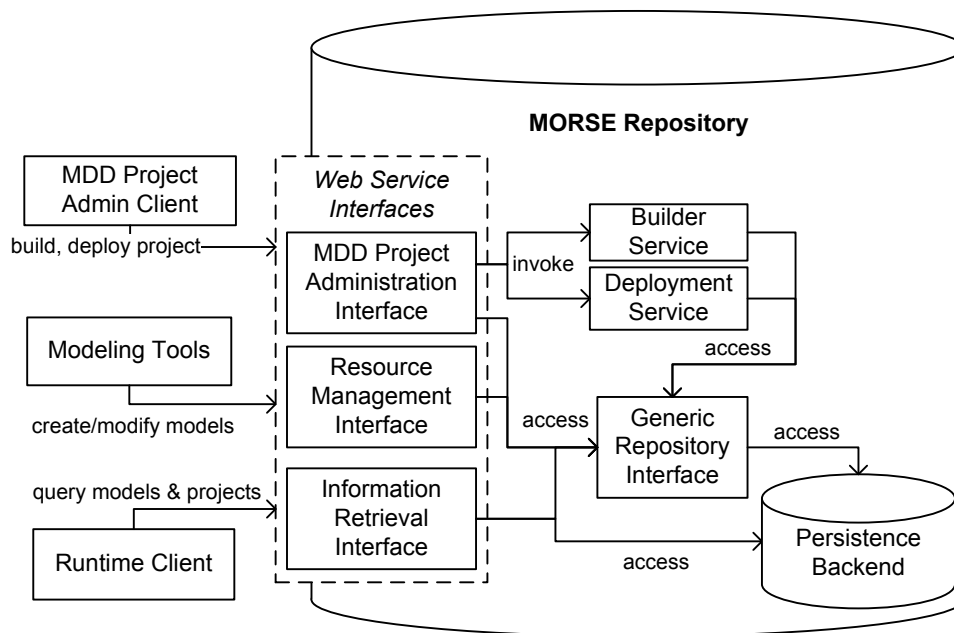


Figure 19 Architecture of the MORSE repository

Because all MDD projects and artefacts are managed in a common model repository, model-aware services can query these for any information on themselves and other model-driven components. Although models and model relations may evolve over time, using UUIDs, it is always possible to retrieve a specific version of a MORSE object. Derived versions, e.g., new versions of the model that were created after deployment time, can easily be identified, permitting a model-aware service to e.g., retrieve and work with the latest version of a model. MORSE can also be beneficial for MDD tools, e.g., in a distributed, collaborative development environment [BB03] while fostering service-orientation to support the MDD design-time tooling. In this case, not the model-aware services or components monitoring them would retrieve and change the models, but the MDD tools such as a model-driven generator. In this section, we do not go into more details about the use scenario, as we want to concentrate on the case of using model-aware services at runtime. However, the MORSE tools themselves use the repository in this way. In Section 4.2.4.i we will illustrate the use of MORSE (see Figure 23).

### 4.2.3. Model repository

The model repository is the main component of MORSE and has been designed with the goal to abstract from specific technologies. Thus, while concepts are taken from e.g., the Unified Modeling Language [ISO05] and also version control systems, MORSE is particularly agnostic to specific modeling frameworks or technologies. The MORSE repository manages objects (MObject) such as projects (MProject) and artefacts (MArtifact) as shown in Figure 20 and Figure 21(a). Additional MORSE object types (explained below) are shown in Figure 21(b). All MObjects are identifiable by UUIDs and can be associated with Dublin Core [KB07] metadata such as title, creator, or date. Note that a UUID also uniquely identifies a particular version of a MORSE object. By navigating across the original or modified relations however, previous and derived versions can be identified. Artefacts are used to manage models and model elements (for details see below), model transformations, and MDD workflows. Besides the versioning of these, the repository supports branching (MBranch) and tagging (MTag) of projects. Note that artefacts can be shared by multiple projects as they can be associated by different tags and branches. They can be changed independently and merged later on.

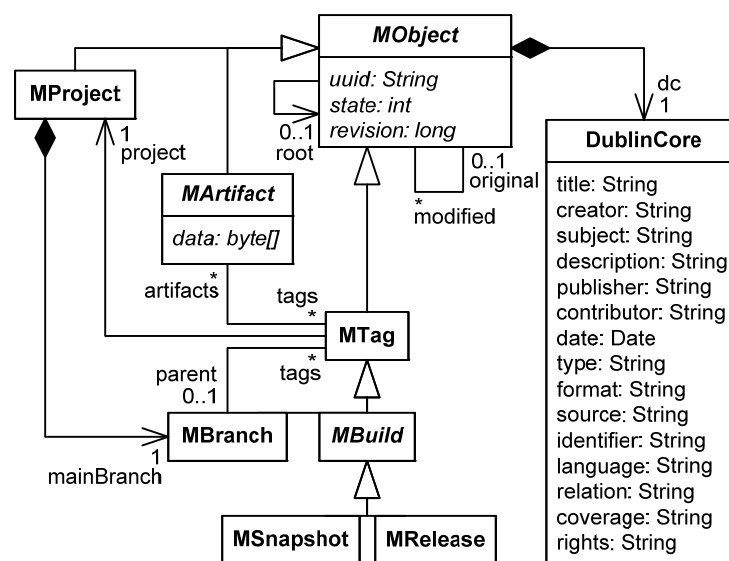


Figure 20 MORSE objects and projects

Typical MDD projects consist of models, transformations, and workflows. An example of a MDD development framework that works with these artefacts is openArchitectureWare (oAW) [Eclipse09]. We have adopted these artefact types and support them in MORSE as shown in Figure 21(a).

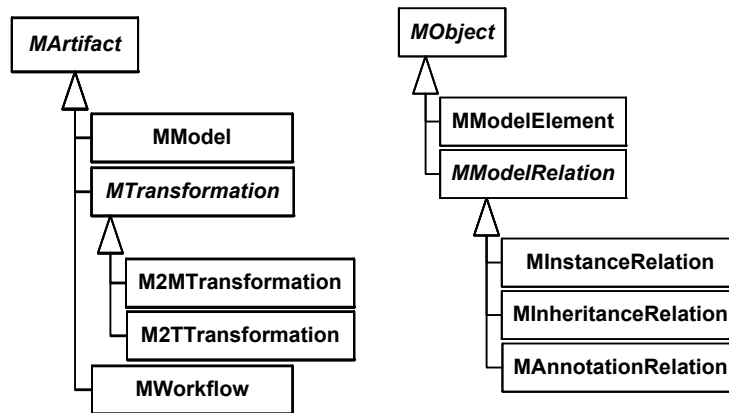


Figure 21 MDD artefacts (left part a) and additional MORSE objects (right part b)

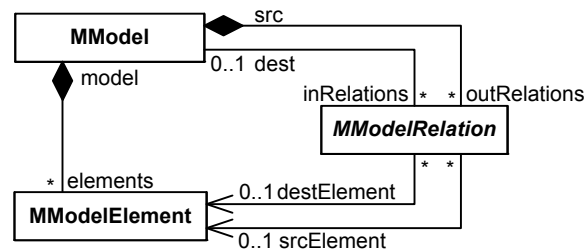


Figure 22 Model element and model relations

Besides the general management of MDD artefacts, MORSE particularly realizes support for models. Models typically contain model elements and have relationships to other models. By capturing and keeping track of these, MORSE facilitates reflection on models, model elements, and model relations. Figure 22 illustrates MModel, MModelElement, and MModelRelation classes that represent these concepts. All these classes derive from MObject and are identifiable and versionable as such. Moreover, MModels are MArtifacts and can use the data attribute to save a serialized form of a model. Examples of different relations are instance-of, inheritance, and annotation relations as shown in Figure 21(b). A model relation (MModelRelation) has a source (src) and destination (dest) model, e.g., an annotated model is the destination model of a MAnnotationRelation and the annotation model depicts the corresponding source model. Besides referring to the models, a model relation may also specify actual model elements (srcElement and destElement). While it would be possible to further specify details, e.g., of model elements, the presented concepts are sufficient for our purposes, i.e., to make models and model elements identifiable and to capture dependencies between different models as introduced through their relations. The models that are stored and versionized within MModels can be retrieved in their serialized form and can further be processed by technology-specific tools, e.g., for introspection, model transformation, or model checking. For the presented classes and concepts, the model repository exposes different services as indicated in Figure 19. Besides an administrative and resource management interface, the repository particularly offers an information retrieval interface to

model-aware services. The information retrieval interface also allows clients to pass complex queries to the persistence backend, e.g., for retrieving all artefacts of a certain branch that have been modified after a certain date. By permitting queries to be passed to and executed at the persistence backend, multiple interactions with the repository can be avoided, resulting in higher performance.

#### 4.2.4. Model-aware services

Services can interact with the MORSE repository at runtime and can profit from its reflective functionalities. We call services that interact with MORSE model-aware. Below we illustrate how they can interact with the repository and what information they may expose. Also we will show how model-aware services can be used by other services and how they can be created.

##### 4.2.4.i. Interaction with the repository

Model-driven, model-aware services can retrieve the MORSE objects from which they have been generated. This is achieved by embedding the UUIDs of the objects into the services, such as the UUID of the build together with UUIDs of corresponding models, model elements, or transformations. At runtime, the service can access these UUIDs and retrieve the MORSE objects from the repository. The model-aware service typically reflects on the information and applies some logic for its further execution or uses the information for performing model transformations.

Figure 23 illustrates a sequence diagram of a model-aware service. After a project and MDD artefacts have been created and checked into the repository, a build for the project is initiated by a client. The builder service retrieves the artefacts and generates a model-aware service, weaving corresponding UUIDs into the code for traceability. Afterward, it is deployed to a Web service framework. Next, a client invokes the service and causes it to interact with the repository. For the models it has been generated from, it needs e.g., to discover and consider new model relations such as new annotations. Therefore it passes the embedded UUIDs of its models to the information retrieval interface of the model repository and requests for the model relations (`getMModel.inRelations`). These are then retrieved and evaluated. Relevant model relations are identified and the related models are requested. Finally, the models are processed and a response is issued to the client.

Please note that the MORSE builder in Figure 23 uses the MORSE repository to obtain the models of the model-aware services in order to generate code for them. This sequence diagram hence illustrates how an MDD tool (in this case the generator of MORSE) can make use of the MORSE architecture in the same way as other components querying models, such as monitoring components.

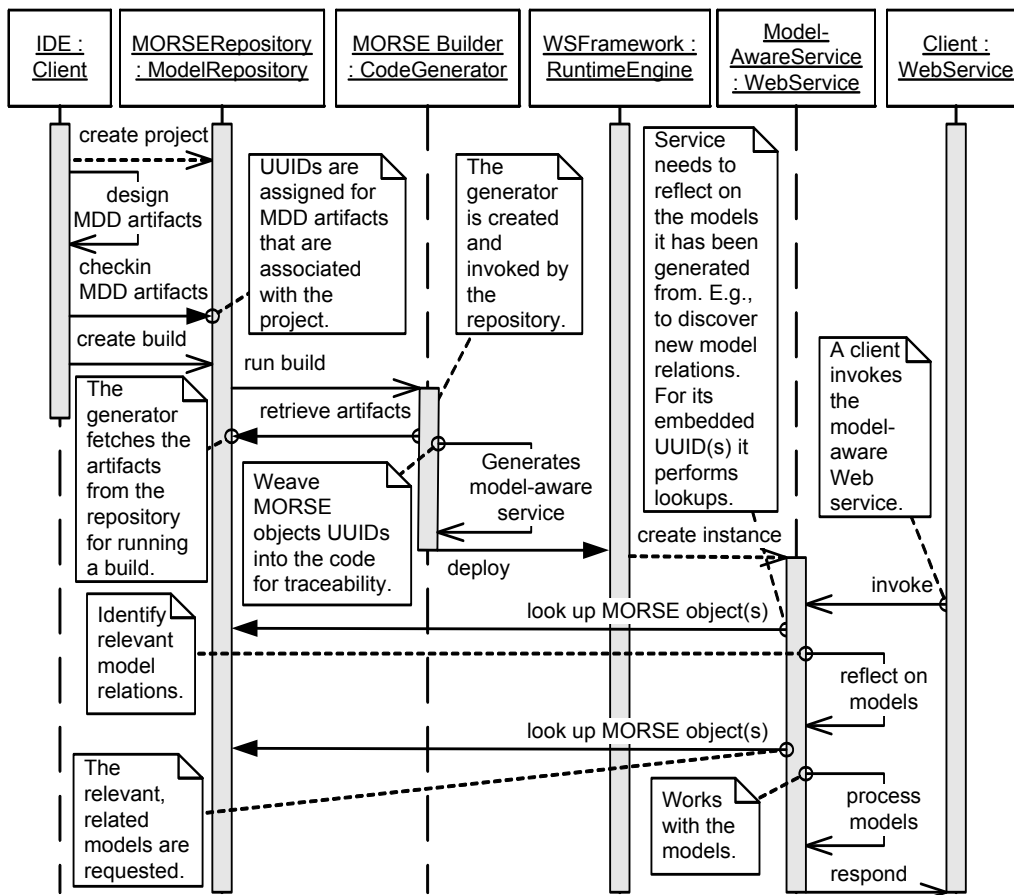


Figure 23 Sequence diagram of a model-aware service

MORSE is publicly available and accessible over Web Services or RESTful Services as shown in Table 1. One service for the management of each artefact within the MORSE repository has been created.

Web Services	RESTful Services
<a href="http://compas-ict.eu/ws/morse/CRequest">http://compas-ict.eu/ws/morse/CRequest</a>	<a href="http://compas-ict.eu/rs/morse/CRequest">http://compas-ict.eu/rs/morse/CRequest</a>
<a href="http://compas-ict.eu/ws/morse/Control">http://compas-ict.eu/ws/morse/Control</a>	<a href="http://compas-ict.eu/rs/morse/Control">http://compas-ict.eu/rs/morse/Control</a>
<a href="http://compas-ict.eu/ws/morse/CRisk">http://compas-ict.eu/ws/morse/CRisk</a>	<a href="http://compas-ict.eu/rs/morse/CRisk">http://compas-ict.eu/rs/morse/CRisk</a>
<a href="http://compas-ict.eu/ws/morse/CRequirement">http://compas-ict.eu/ws/morse/CRequirement</a>	<a href="http://compas-ict.eu/rs/morse/CRequirement">http://compas-ict.eu/rs/morse/CRequirement</a>
<a href="http://compas-ict.eu/ws/morse/CSource">http://compas-ict.eu/ws/morse/CSource</a>	<a href="http://compas-ict.eu/rs/morse/CSource">http://compas-ict.eu/rs/morse/CSource</a>
<a href="http://compas-ict.eu/ws/morse/EnumWeight">http://compas-ict.eu/ws/morse/EnumWeight</a>	<a href="http://compas-ict.eu/rs/morse/EnumWeight">http://compas-ict.eu/rs/morse/EnumWeight</a>
<a href="http://compas-ict.eu/ws/morse/EnumComplianceSource">http://compas-ict.eu/ws/morse/EnumComplianceSource</a>	<a href="http://compas-ict.eu/rs/morse/EnumComplianceSource">http://compas-ict.eu/rs/morse/EnumComplianceSource</a>
<a href="http://compas-ict.eu/ws/morse/EnumStandardFramework">http://compas-ict.eu/ws/morse/EnumStandardFramework</a>	<a href="http://compas-ict.eu/rs/morse/EnumStandardFramework">http://compas-ict.eu/rs/morse/EnumStandardFramework</a>
<a href="http://compas-ict.eu/ws/morse/EnumPreventiveDetective">http://compas-ict.eu/ws/morse/EnumPreventiveDetective</a>	<a href="http://compas-ict.eu/rs/morse/EnumPreventiveDetective">http://compas-ict.eu/rs/morse/EnumPreventiveDetective</a>
<a href="http://compas-ict.eu/ws/morse/EnumStandardKey">http://compas-ict.eu/ws/morse/EnumStandardKey</a>	<a href="http://compas-ict.eu/rs/morse/EnumStandardKey">http://compas-ict.eu/rs/morse/EnumStandardKey</a>
<a href="http://compas-ict.eu/ws/morse/EnumEventbasedPeriodic">http://compas-ict.eu/ws/morse/EnumEventbasedPeriodic</a>	<a href="http://compas-ict.eu/rs/morse/EnumEventbasedPeriodic">http://compas-ict.eu/rs/morse/EnumEventbasedPeriodic</a>
<a href="http://compas-ict.eu/ws/morse/EnumTimeUnit">http://compas-ict.eu/ws/morse/EnumTimeUnit</a>	<a href="http://compas-ict.eu/rs/morse/EnumTimeUnit">http://compas-ict.eu/rs/morse/EnumTimeUnit</a>
<a href="http://compas-ict.eu/ws/morse/EnumVersioningActivity">http://compas-ict.eu/ws/morse/EnumVersioningActivity</a>	<a href="http://compas-ict.eu/rs/morse/EnumVersioningActivity">http://compas-ict.eu/rs/morse/EnumVersioningActivity</a>
<a href="http://compas-ict.eu/ws/morse/CRule">http://compas-ict.eu/ws/morse/CRule</a>	<a href="http://compas-ict.eu/rs/morse/CRule">http://compas-ict.eu/rs/morse/CRule</a>
<a href="http://compas-ict.eu/ws/morse/EnumCRuleType">http://compas-ict.eu/ws/morse/EnumCRuleType</a>	<a href="http://compas-ict.eu/rs/morse/EnumCRuleType">http://compas-ict.eu/rs/morse/EnumCRuleType</a>
<a href="http://compas-ict.eu/ws/morse/EnumPolicyType">http://compas-ict.eu/ws/morse/EnumPolicyType</a>	<a href="http://compas-ict.eu/rs/morse/EnumPolicyType">http://compas-ict.eu/rs/morse/EnumPolicyType</a>
<a href="http://compas-ict.eu/ws/morse/AtomicCRule">http://compas-ict.eu/ws/morse/AtomicCRule</a>	<a href="http://compas-ict.eu/rs/morse/AtomicCRule">http://compas-ict.eu/rs/morse/AtomicCRule</a>
<a href="http://compas-ict.eu/ws/morse/CompositeCRule">http://compas-ict.eu/ws/morse/CompositeCRule</a>	<a href="http://compas-ict.eu/rs/morse/CompositeCRule">http://compas-ict.eu/rs/morse/CompositeCRule</a>
<a href="http://compas-ict.eu/ws/morse/UnaryCRule">http://compas-ict.eu/ws/morse/UnaryCRule</a>	<a href="http://compas-ict.eu/rs/morse/UnaryCRule">http://compas-ict.eu/rs/morse/UnaryCRule</a>

<a href="http://compas-ict.eu/ws/morse/BinaryCRule">http://compas-ict.eu/ws/morse/BinaryCRule</a>	<a href="http://compas-ict.eu/rs/morse/BinaryCRule">http://compas-ict.eu/rs/morse/BinaryCRule</a>
<a href="http://compas-ict.eu/ws/morse/BinaryOperator">http://compas-ict.eu/ws/morse/BinaryOperator</a>	<a href="http://compas-ict.eu/rs/morse/BinaryOperator">http://compas-ict.eu/rs/morse/BinaryOperator</a>
<a href="http://compas-ict.eu/ws/morse/TechControl">http://compas-ict.eu/ws/morse/TechControl</a>	<a href="http://compas-ict.eu/rs/morse/TechControl">http://compas-ict.eu/rs/morse/TechControl</a>
<a href="http://compas-ict.eu/ws/morse/Fragment">http://compas-ict.eu/ws/morse/Fragment</a>	<a href="http://compas-ict.eu/rs/morse/Fragment">http://compas-ict.eu/rs/morse/Fragment</a>

**Table 1 Overview of MORSE availability over Web services and RESTful services**

#### 4.2.4.ii. Informational operations

We have seen how model-aware services can interact with the repository, e.g., after invocation. Besides this, model-aware services may also offer information on them to other services (cf. Figure 18), i.e., disclose the MORSE objects they have been created from or are related to. For such model-aware services, we propose the operations displayed in Table 2. The parameter passed to the operations specifies the type of the MORSE object in question, such as MBuild, MProject, or MModel. As a result, the service returns MORSE objects of the respective type. For example, in order to retrieve the models from which the model-aware service has been generated the getMObject operation is called with the parameter MModel. With the proposed operations it is possible to interrogate a model-aware service for its build, originating project, and MDD artefacts.

Return Type	Operation	Parameter
MObject[]	getMObject	type of MObject
UUID[]	getMObject.uuid	type of MObject

**Table 2 Model-aware service operations**

#### 4.2.4.iii. Integrating with model-aware services

A service or process may integrate with and use model-aware services (see also Figure 18 in Section 4.2.2). A Model-aware service can support model-driven systems in the sense that it can look-up and work with the MDD artefacts they have been generated from. During runtime, it receives events from these systems that contain MORSE identifiers and queries the model repository. The BPEL extension for the MORSE traceability is described in [D4.2].

### 4.3. Integration of Fragmento and Morse

In this section we describe, how the model repository MORSE is integrated with Fragmento. The usage of “proxy objects” allows linking the process fragments realizing a particular technical compliance control to their source, e.g., to a particular section in an electronic version of a legal document. Maintaining such a link is essentially important in order to provide traceability, and in order to be able to flexibly react on changes in laws. Changes in laws may subsequently require changes of the technical compliance controls that realize them, thus establishing a link between them is helpful in many scenarios.

The integration of the two repositories is actually straightforward: For each fragment that is stored in Fragmento we are creating a proxy object in MORSE. Basically we have created a one-way synchronization driven by an external component which can either be executed on demand or according to a predefined schedule (e.g., as cronjob). The types of artefacts, which are synchronized, are limited to process fragments. Our work with the use cases has proven that this limited to the objects synchronization is sufficient.

For this synchronization component we have used the model-driven code generation facilities provides by the Axis 2 Web service framework [Apache09b]. The code generator has been used to create a Java skeleton for invoking the MORSE repository, along with a Java skeleton

for invoking Fragmento. The algorithm that implements the synchronization of the repositories is sketched in Listing 3. For providing easy access to the content ‘behind’ the proxy object we store a URL under which the latest version of the actual item can be retrieved using a HTTP/Get request, which can be realized by clicking a link in a browser.

```
void Synchronize (Date start, Date end, String type){
    //retrieve a list of artefacts created within a particular interval
    List<Item> items = fragmentService.browse (start, end, type);
    if (items.length > 0){
        for (int i = 0; i <= items.length; i++){
            //For each new item: create proxy object
            modelService.createFragmentProxy (item.id, item.url);
        }
        System.out.println("Synchronization complete!");
    }
    else {
        System.out.println("Synchronization complete, no new items");
    }
}
```

**Listing 3 Synchronization of new artefacts**

## 5. Process generation tool prototype description

The process generation tool described in this deliverable is actually not used as a separate component, but as an integral part of the VbMF developed in WP1 (cf. [D1.2, D1.3]). This MDSF framework supports the task of process generation by the means of code generators. The code generators produce schematic process descriptions and service descriptions of processes. Moreover, deployment configurations that can be used to deploy the generated processes into the process engine are also generated. This allows focusing more on process models, rather than on technical details. In this deliverable, we briefly summarize the main points in designing and implementing the code generation for the process artefacts.

COMPAS project contributes not only to enhance the automation in compliance engineering but also to enhance and accelerate business process development. The MDSF Software Framework [D1.2, D1.3] developed in WP1 supports this task by the means of code generators. On the one hand, the code generators produce schematic process descriptions and service descriptions of processes. Moreover, deployment configurations that can be used to deploy the generated processes into process engines will be also generated. Thus, the stakeholders can focus more to process models rather than involving in such technical details. On the other hand, static model validations and documentation generators can be employed in the code generators to perform checking at design time along with creating relevant documentations of process designs and implementations.

## 5.1. Requirements and specification

The MDSD Software Framework provides process view models at two levels of abstraction. Abstract view models are used for representing business- and domain-oriented concepts and knowledge whilst the technology-specific view models are leveraged for aligning these concepts with the IT infrastructure. In COMPAS, we use the combination of BPEL [OASIS07], WSDL [W3C01], and XML [W3C08] as representative languages for process descriptions and service descriptions because these are de facto standards, widely used in industry for business process development. Hence, the process generation tools, i.e., code generators, have to transform concepts of the technology-specific views models into process descriptions in BPEL and service descriptions of processes in WSDL and XML schema. In addition, the Apache ODE engine [Apache09a] is chosen as process engine to deploy and enact COMPAS use cases because it is open-source and compliant with the WSBPEL 2.0 standard. Thus, we also develop code generators to produce configurations for automatically deploying business processes in Apache ODE.

## 5.2. Design and implementation

The code generators are parts of MDSD Software Framework, and therefore, are elaborated in [D1.2]. In this deliverable, we briefly summarize the main points in designing and implementing of the code generators.

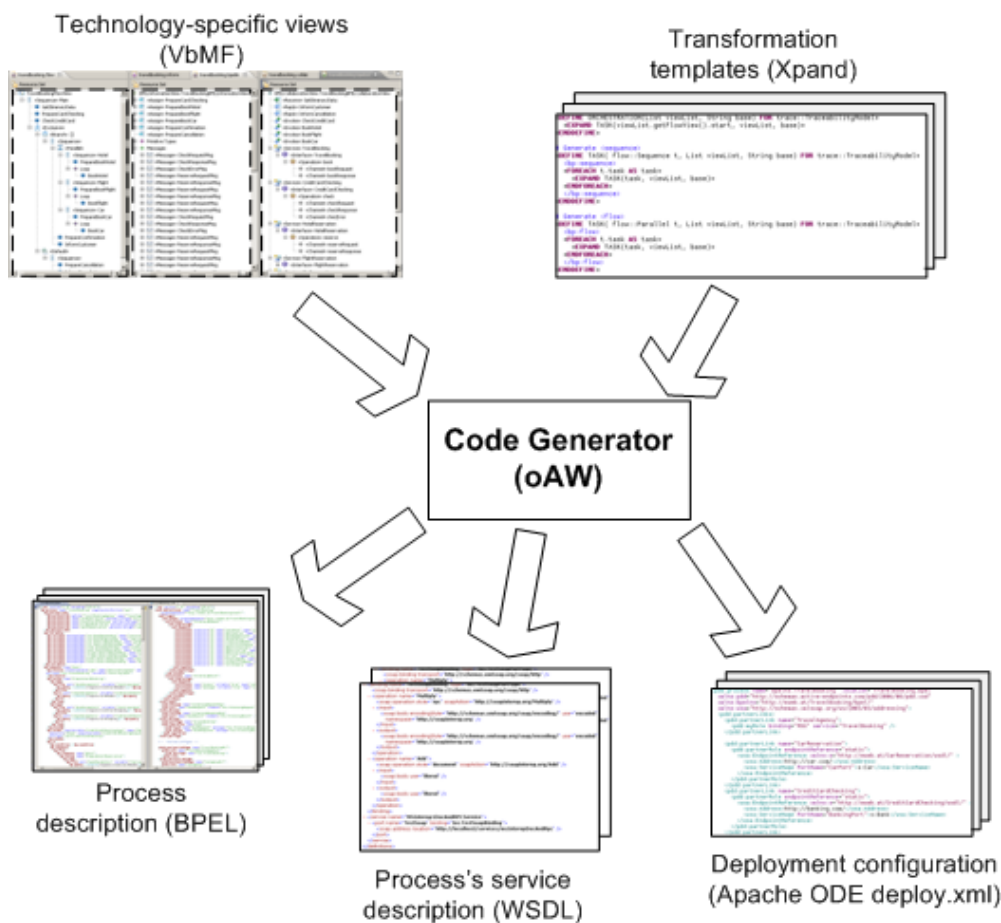


Figure 24 Overview of the process generation tool

The view models are developed in the MDS Software Framework which is based on View-based Modeling Framework (VbMF). We have implemented the code generators using the template-based techniques provided by the Xpand language of the openArchitectureWare framework [Eclipse09]. The code generators take the view models and templates as inputs and produce BPEL descriptions representing the functionality of the process, WSDL descriptions representing the standard interfaces that the process exposes to its consumers, and the *deploy.xml* describing the configurations for deploying and enacting the process in Apache ODE engine.

## 6. Generation of Business Protocols

Business Protocol Monitoring tool [D5.4] uses a simple abstraction in order to specify the monitorable properties, which are written in BPath language. This abstraction is known as Business Protocol. The purpose of the business protocol abstraction is essentially to specify the set of conversations (sequence of messages) that are supported by a business process [BCT04]. Formally, a business protocol can be defined as a tuple  $P = (S, s_0, F, M, T)$  where:

- $S$  is a finite set of states the process goes through during its execution
- $M$  is a set of messages.
- $s_0$  is the initial state
- $F$  represents the finite set of final states.
- $T \subseteq S \times S \times M \times \{+, -\}$  is the set of transitions, where every transition is labelled with a message name and its polarity, when a message is consumed by the protocol, the transition is assigned the polarity sign(+), and when it is produced by the protocol, the transition is assigned the sign (-).

In the following, we present a way to generate a business protocol specification from a business process specified by means of BPEL. Then, indirectly, from BPMN to business protocol by using existing algorithms to transform BPMN process to BPEL process such as [ODH06].

BPEL is an XML-based language for specifying process behaviour by defining the activities it is composed of and the external processes that interact with it. BPEL activity can be basic or structured. Basic activities can be communication activities ensuring communication functions like sending, receiving messages, or those doing internal tasks like assigning a variable or throwing an exception. Structured activities prescribe the order in which a collection of activities take place, like activities should be executed in sequence or in a loop. A structured activity may contain other activities, and even structured activities.

In order to transform an XML BPEL process to a business protocol, we propose to consider the following steps:

- (1) Remove all elements from the XML BPEL process, except communication and structured activities, and let  $xBpel$  the resulting XML tree.
- (2) Build an object representation  $oBpel$  corresponding to  $xBpel$ , where:
  - a. Each activity  $a$  of a type  $T$  in  $xBpel$  is represented as an instance  $a$  of a class of type  $T$ , each class defines two properties: input, and output, and a Merge() method (the details of each class type will be presented later).
  - b. If  $b$  is a child node of  $a$  in  $xBpel$ , then the activity instance  $b \in$ 
    - a. children.
- (3) The protocol automaton (message-based automaton) will be generated as defined in the following function, with  $oBpel$  as an argument:

```

Protocol    Begin
(a):        Let P= ( {s0, f0}, s0, {f0}, {}, {} ) an
            automaton;
            For each b in a.children do
                Begin
                    b.Merge() ;
                    Replace a.input by s0 in P;
                    Replace a.output by f0 in P;
                End
            return P;
        End

```

In the following we present in details how each activity is represented in BPEL, and how input, output and Merge () method of it corresponding activity instance in *oBpel* are calculated.

## 6.1. Communication activities

### 6.1.1. Receive activity

This activity does a blocking wait for a matching message to arrive, it is encoded in BPEL as:

```
<receive name="ai" partnerLink="pl" portType="pt" operation="op"
        variable="var">
```

Merge():

Begin

- Let  $s_1, s_2$  be two new states ;
- Let a message  $m_1=(pl, op, inMessage)$ ; //inMessage is the input message of the operation op, which can be extracted by parsing the wsdl file defining the messages involved in the BPEL process definition.
- Let  $t_1/\{t_2=(s_1, s_2, m_1, +)\}$  be a transition;
- Add  $s_1, s_2, m_1, t_1$  to P;
- Set input, and output properties to  $s_1$ , and  $s_2$  respectively;

End

### 6.1.2. Reply activity

Send a message in response to a previously received message; it is encoded in BPEL as:

```
<reply name="ai" partnerLink="pl" portType="pt" operation="op"
        variable="var">
```

Merge():

Begin

- Let  $s_1, s_2$  be two new states , message  $m_1=(pl, op, outMessage)$ , where *outMessage* is the output message of the operation *op*;

- Let  $t_1/\{t_2=(s_1,s_2,m_1,-)\}$  be a transition;
  - Add  $s_1,s_2,m_1,t_1$  to P;
  - Set input, and output properties to  $s_1$ , and  $s_2$  respectively;
- End

### 6.1.3. Invoke activity

Invoke a one-way or request response operation, represented in BPEL as:

```
<invoke name="ai" partnerLink="pl" portType="pt" operation="op"
        inputVariable="inVar" outputVariable="outVar"/>
```

Merge():

Begin

- Let  $s_1,s_2,s_3$  be new states ;
  - Let message  $m_1=(pl,op,outMessage)$ , where *outMessage* is the output message of the operation *op*;
  - Let a message  $m_2=(pl,op,inMessage)$ , where *outMessage* is the input message of the operation *op*;
  - Let  $t_1/\{t_2=(s_1,s_2,m_1,-)\}$  be a transition;
  - Let  $t_2/\{t_2=(s_2,s_3,m_2,+)\}$  be a transition;
  - Add  $s_1,s_2,m_1,t_1,t_2$  to P;
  - Set input, and output properties to  $s_1$ , and  $s_3$  respectively;
- End

## 6.2. Structured activities

### 6.2.1. If activity

Allows selecting exactly one branch of activity from a set of choices.

```
<if name="ai" >
  <condition .../>
  activity1
  elseactivity1
  elseactivity2
  ...
</if>
```

Input, output properties, and Merge () method of an **If activity** instance are defined as follows:

$input = \cup a.input / a \in children$  (the set of input of all children activities)

$output = \cup a.output / b \in children$  (the set of output of all children activities)

Merge() :

Begin

- For each  $a \in children$  do:
  - a.Merge();

End

### 6.2.2. Sequence activity

States that contained activities should be performed sequentially in lexical order.

```
<sequence name="ai" >
  activity1
  activity2
  ...
</sequence>
```

Input, output properties, and Merge () method of a sequence activity instance is defined as follows:

input= a.input, where *a* the first child.

output= a.output, where *a* the last child.

Merge ():

Begin

- Merge children activities like with *If activity*
- For each *a, b* children activities, and *b* the following sibling of *a*

Begin

Let *s* be a new state

Add *s* to P

Replace *a.output*, and *b.input* by *s* in P

End

End

### 6.2.3. RepeatUntil activity

Allows a single sub activity, which is repeated until a predicate holds. It is encoded as:

```
<repeatUntil name="ra" >
  <condition .../>
  activity1
</repeatUntil>
```

We define the *input* and *output* properties of a repeatUntil by :

input=a.input / a the first sub-activity of *a<sub>i</sub>*,

output= input

Merge ():

Begin

- Merge children activities like with a *Sequence activity*;
- Let *s* be a new state;
- Add *s* to P;
- Let *b* be the last child activity;
- Replace *input*, and *b.output* by *s* in P;

End

## 6.2.4. Flow activity

Contained activities are executed in parallel, it is encoded as:

```
<flow name="fa" >
  activity1
  activity2
  ...
</ flow >
```

Input and output will be determined after calculating the product of business protocols obtained from each child instance activity.

Merge():

Begin

- Let  $a, b, \dots$  be children instance activities;
- Let  $P_x = \text{Protocol}(a) \times \text{Protocol}(b), \dots$ ;
- Set  $\text{Input} = s / s$  the initial state of  $P_x$ ;
- Set  $\text{Output} = s / s \in P_x, s \in F_x$  (there is only one final state in  $P_x$ );

End

## 7. Roles of prototypes and usage in the COMPAS usage scenarios

In this section we explain the role of the COMPAS prototypes described in this deliverable and their application to both the ICT security scenario (Section 7.1) and the Advance telecom service for Mobile Virtual Network Operators scenario “WatchMe” (Section 7.2).

### 7.1. The ICT security scenario

The prototypes described in this deliverable play a major role in the realization of the usage scenarios defined in COMPAS (cf. [D6.1], [D6.3]). In the THALES use case (i.e. the ICT Security Scenario) a process has to be augmented with activities and checks related to compliance. In the first step a process designer queries the MORSE, using the CRLT developed in WP2 (cf. [D2.6]). For instance the query could be, to search for compliance requirements related to Basel II [BASELII] which are related to loan processing.

As a result the process designer finds a set of compliance requirements, which are associated with formal rules (for verification), technical annotations (for execution and monitoring) and compliance fragments (for integration into the process). In the next step the process designer can choose to check his existing process against the formal rules associated with his query results. Therefore, the CRLT invokes the verification functionality developed in WP3 for checking the process against the formal rules. If all rules are satisfied then the process is already (design time) compliant and can be executed on the process engine.

However, if rules are violated this indicates that the process is not designed in compliance with the requirements. In case of a violation the process designer has to re-design the process by integrating compliance fragments or by rearranging, removing or changing the process structures which caused the violation. All related artefacts which are required for proper implementation of a compliance requirement are retrieved from the fragment repository

Fragmento. The compliance fragments are manually integrated into the process. After integration, the process designer may check the augmented process again, for having proof that all requirements are now satisfied and all fragments have been integrated in a correct manner.

The process (which is now compliant by design) can then be deployed on the process engine for execution. Also the technical annotations for execution and monitoring have to be deployed into the infrastructure for enabling runtime checking. As not all requirements can be checked during design time, the process engine emits execution events which are evaluated (based on the technical annotations) in the monitoring infrastructure developed in WP5. These events carry particular identifiers, which allow tracing a detected violation back to the models for business intelligence.

## **7.2. Advance telecom service for Mobile Virtual Network Operators**

The compliance requirements of the MVNO scenario “WatchMe” cover the areas licensing, quality of service and internal policies. The corresponding DSLs developed within COMPAS can be used within DSL editors and the DSL Transformation components in order to create and express the compliance requirements on a technical level. For the list of compliance requirements of the WatchMe scenario see Table 4 within COMPAS deliverable [D5.4].

DSL model instances specify compliance requirements and serve as input for the VbMF. The VbMF is used to create the different view models to describe and define both, the business and the technical aspects of the WatchMe business process. Afterwards, the DSL model instances as well as the view models are stored within the Model Repository.

The view models serve as an input for the Code Generators within the MDSD software framework, which are responsible for generating BPEL process models including the information required for later deployment on the extended Apache ODE process engine. After the generation, the process model represented in BPEL is stored within the fragment repository, which synchronizes with the Model Repository automatically.

Once the compliance requirements and view models are stored in the Model Repository and the corresponding generated process model is stored in Fragmento, the static compliance check at design time can be performed, analogously to the ICT security scenario.

During the execution of WatchMe process instances, the process engine emits events containing the traceability information to the compliance architecture part of the COMPAS overall architecture via the Enterprise Service Bus.

## **8. Prototype websites**

The websites with further documentation of the described COMPAS components are linked at <http://compas-ict.eu/prototypes.php>.

## **9. Reference documents**

### **9.1. Internal documents**

[DoW] “Description of Work” for COMPAS, Version 15 of 2007-09-28.

- [D1.2] “Core meta-models, transformation templates, and languages”, Version 1.0 of 2009-12-31.
- [D1.3] “MDSD Software Framework for Business Compliance – Final Version”, Version 2.0 of 2010-12-31
- [D1.4] “Runtime Environment”, Version 1.0 of 2010-12-23
- [D2.6] “Implementation of an Integrated Prototype handling Interactive User Specified Compliance Requests in a Compliance Language”, Version 1.0 of 2009-12-31.
- [D4.1] “State-of-the-art report on the existing approaches to improving reusability of processes and service compositions”, Version 2.0 of 2009-06-08.
- [D4.2] “BPEL extensions for compliant services”, Version 1.0 of 2009-12-31.
- [D4.3] “Classification and specification of reusable process artefacts”, to be published in M30 of the COMPAS project duration.
- [D4.4a] “Supporting infrastructure – process engine, process artefact repository, process generation tool, Version 1.0 of 2009-12-22.
- [D5.4] “Reasoning mechanisms to support the identification and the analysis of problems associated with user requests”, Version 2.0 of 2009-12-31.
- [D6.1] “Use Case, Metrics and Case Study”, Version of 1.0 2008-07-31.
- [D7.1] “Public Web-Site”, <http://www.compas-ict.eu>

[All links were last followed on December 17, 2010.]

## 9.2. External documents

- [Apache04] Apache Software Foundation: Apache License Version 2.0, 2004, <http://www.apache.org/licenses/LICENSE-2.0.html>.
- [Apache09a] Apache Software Foundation: Apache ODE (Orchestration Director Engine), <http://ode.apache.org>.
- [Apache09b] Apache Software Foundation: Apache Axis2, <http://ws.apache.org/axis2/>
- [Apache09c] Apache Software Foundation: Apache Tomcat, <http://tomcat.apache.org/>.
- [BASELII] Basel Committee on Banking Supervision: International Convergence of Capital Measurement and Capital Standards, 2006.
- [BB03] G. Booch, A. W. Brown, Collaborative development environments, Advances in Computers, vol. 59, pp. 1 – 27, 2003.
- [BCBS06] Basel Committee on Banking Supervision: Basel II: International Convergence of Capital Measurement and Capital Standards, A Revised Framework Comprehensive Version, June 2006, <http://www.bis.org/publ/bcbs128.pdf>.
- [BCT04] B. Benatallah, F. Casati, and F. Toumani, “Web Service Conversation Modeling: A Cornerstone for E-Business Automation,” IEEE Internet Computing, vol. 8, 2004, pp. 46-54.
- [Display09] Displaytag Team: Display tag library, <http://displaytag.sourceforge.net/>.

- [Eclipse09] openArchitectureWare working group at Eclipse.org, oAW is now part of the Eclipse Modeling Project, 2009, <http://www.eclipse.org/workinggroups/oaw/>.
- [ESPER09] EsperTech: Esper, 2009, <http://esper.codehaus.org>.
- [FCD+09] F. Daniel, F. Casati, V. D'Andrea, S. Strauch, D. Schumm, F. Leymann, E. Mulo, U. Zdun, S. Dustdar, S. Sebahi, F. de Marchi, M. Hacid: Business Compliance Governance in Service-Oriented Architectures. In: Proceedings of the IEEE Twenty-Third International Conference on Advanced Information Networking and Applications (AINA'09), 2009.
- [Frgmto10] Fragmento – Fragment-oriented Repository. Online Documentation, 2010. <http://www.iaas.uni-stuttgart.de/forschung/projects/fragmento/start.htm>
- [HZD09] T. Holmes, U. Zdun and S. Dustdar: MORSE: A Model-Aware Service Environment, In: Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference (APSCC'09), IEEE Computer Society Press, 2009.
- [ISO05] ISO, ISO/IEC 19501:2005 Information technology – Open Distributed Processing – Unified Modeling Language (UML), v1.4.2, 2005, <http://www.omg.org/cgi-bin/doc?formal/05-04-01>.
- [ITU04] International Telecommunication Union, ISO/IEC 9834-8 Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components, 2004, <http://www.itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf>.
- [KB07] J. Kunze and T. Baker, “The Dublin Core Metadata Element Set,”, The Internet Engineering Task Force, Request for Comments, 2007, <http://www.ietf.org/rfc/rfc5013.txt>
- [KKL07] R. Khalaf, D. Karastoyanova, F. Leymann: Pluggable Framework for Enabling the Execution of Extended BPEL Behavior. In: Proceedings of the 3<sup>rd</sup> International Workshop on Engineering Service-Oriented Application (WESOA'2007), Springer, 2007.
- [KKS+06] D. Karastoyanova, R. Khalaf, R. Schroth, M. Paluszek, F. Leymann: BPEL Event Model, Technical Report 2006/10, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany.
- [MASTER09] MASTER Project Deliverable D2.3.1: Technical architecture and APIs for single trust domain, 2009, [http://www.master-fp7.eu/index.php?option=com\\_docman&task=doc\\_download&gid=21&Itemid=60](http://www.master-fp7.eu/index.php?option=com_docman&task=doc_download&gid=21&Itemid=60).
- [ML09] Z. Ma, F. Leymann: BPEL Fragments for Modularized Reuse in Modeling BPEL Processes. In: Proceedings of the 5<sup>th</sup> International Conference on Networking and Services (ICNS), IEEE Computer Society, 2009.
- [OASIS07] OASIS. Web Services Business Process Execution Language, version 2.0, Apr 2007, <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>.

- [ODH06] C. Ouyang, M. Dumas, and A.H.M.T. Hofstede: From BPMN Process Models to BPEL Web Services. In: Proceedings of the 4th International Conference on Web Services (ICWS'06), pp. 285—292, IEEE Computer Society, 2006.
- [Pos10] A. Poszlovszki: Process Fragment Recognition and Emphasis. Diploma thesis no. 3001, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2010. [ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart\\_fi/DIP-3001/DIP-3001.pdf](ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-3001/DIP-3001.pdf)
- [Postgre09] PostgreSQL Global Development Group: PostgreSQL, <http://www.postgresql.org/>.
- [Hibernate09] Red Hat: Hibernate, <https://www.hibernate.org/>.
- [SKL+10] D. Schumm, D. Karastoyanova, F. Leymann, S. Strauch: Fragmento: Advanced Process Fragment Library. Proceedings of the 19th Int. Conference on Information Systems Development (ISD 2010), Springer, 2010.
- [SLM+10] D. Schumm, F. Leymann, Z. Ma, T. Scheibler, S. Strauch: Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. In: Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI'10), 2010.
- [SLS10] D. Schumm, F. Leymann, A. Streule: Process Viewing Patterns. Proceedings of the 14th IEEE International EDOC Conference (EDOC 2010), IEEE Computer Society Press, 2010.
- [SoapUI] eviware: SoapUI, May 2010. <http://www.soapui.org/>
- [Spring09] SpringSource: Spring, <http://www.springsource.org/>.
- [Ste08] T. Steinmetz: Ein Event-Modell für WS-BPEL 2.0 und dessen Realisierung in Apache ODE, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, Diploma Thesis 2729, 2008 (in German).
- [W3C01] W3C. Web Services Description Language (WSDL) 1.1, Mar 2001, <http://www.w3.org/TR/wsdl>.
- [W3C08] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition), Nov 2008, <http://www.w3.org/TR/xml/>.
- [XMLO09] XMLO – The eXtensible Markup Language repOsitory, 2009. <http://www.iaas.uni-stuttgart.de/forschung/projects/master/xmlo.php>.

[All links were last followed on November 25, 2010.]