D1.1

Version: 2.0
Date: 2009-06-08
Dissemination status: PU
Document reference: D1.1

Model-driven Integration Architecture for Compliance

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

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

Project no. 215175

COMPAS

Compliance-driven Models, Languages, and Architectures for Services

Specific Targeted Research Project

Information Society Technologies

Start date of project: 2008-02-01 Duration: 36 months

D1.1 Model-driven Integration Architecture for Compliance

Revision 2.0

Due date of deliverable: 2008-12-31

First submission date: 2008-12-30

Latest submission date: 2009-06-08

Organisation name of lead partner for this deliverable:

TUV Technische Universität Wien, AT

Contributing partner(s):

CWI Stichting Centrum voor Wiskunde en Informatica, NL

USTUTT Universitaet Stuttgart, DE

TARC-PL Telcordia Poland, PL

UNITN Universita degli Studi di Trento, IT

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

History chart

Issue	Date	Changed page(s)	Cause of change	Implemented by
0.1	2008-10-30	Table of contents	New document	TUV
0.2	2008-11-18	Initial draft	TUV review	TUV
0.21	2008-11-19		Addressing TUV review	TUV
0.3	2008-12-02		Request for review	TUV
0.4	2008-12-10	Whole document	Addressing reviews	TUV
0.5	2008-12-12	Architecture overview	Added explanations for the relationships of components by partner contributions	TUV
0.6	2008-12-19	Whole document	Addressing reviews	TUV
0.9	2008-12-23	Whole document	Revising	TUV
		Document name	Renamed to comply with Project Quality Plan	TUV
0.98	2008-12-29	Table 1	Updating with partner contributions	TUV
1.00	2008-12-30	Table 1	Updating with partner contributions	TUV
		Whole document	Prepare for releasing	TUV
1.10	2009-05-08	Whole document	Revise according to reviewers' comments	TUV
2.0	2009-06-08		Approval & Release	TUV

Authorisation

No.	Action	Company/Name	Date
1	Prepared	TUV	2008-12-23
2	Approved	TUV	2008-12-27
3	Released	TUV	2008-12-30
4	Prepared	TUV	2009-05-08
5	Approved	TUV	2009-06-08
3	Released	TUV	2009-06-08

Contents

List of figures

List of tables

List of listings

Abstract

Business compliance today is usually implemented on a per case basis. In this deliverable, we present the initial architecture of a framework that aims to implement this compliance in a more generalized manner. The framework leverages the advantages of the model driven software development paradigm to rapidly develop and stably evolve a business compliance solution. We give an overview of the components that make up this framework and how they integrate with each other, as well as explaining how the model driven approach addresses some of the challenges experienced when developing compliance solutions.

1. Introduction

1.1. Purpose and scope

The COMPAS project aims to develop a demonstrable approach to solving the problem of business compliance. For the sake of demonstration, a prototype of a business compliance software framework, based on the model driven development paradigm, is to be developed. WP1 focuses on development of the core and modelling aspects of this business compliance framework, as well as ensuring that it integrates with the components from the different WPs in COMPAS project.

The purpose of this deliverable is to describe the architecture of this framework and elaborate on how it shall interface with the different prototypes from other WPs (2, 3 and 5). It provides a high level overview of the components from all WPs and then provides a more detailed view of the components and explains how the components integrate with each other to realise a business compliance solution.

1.2. Document overview

The deliverable has a number of sections that are now described briefly. Section 2 gives a high level overview of the whole COMPAS architecture. This is followed by Section 3 that describes the components in more detail and explains how they integrate with each other. In Section 3, we introduce in detail the model-driven tool-chain that comes together with concepts of domain specific languages to serve as the basis for COMPAS model-driven integration architecture. Section 4 will examine a number of compliance concerns and present our proposed approaches to modelling these concerns. Finally, Section 5 comes to summarise the main contributions presented in this document.

1.3. Definitions and glossary

Architectural view: a view is a representation of a whole system from the perspective of a related set of concerns.

Service Oriented Architecture (SOA): an architectural style in which software components or software systems operate in a loosely-coupled environment, and are delivered to end-users in terms of software units, namely, services. A service provides a standard interface (e.g., service interfaces described using WSDL), and utilises message exchange as the only communication method.

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

Business compliance (or compliance for short): The goal to ensure that the systems of a company comply with regulatory or legislative provisions, or similar business requirements given through outer influences. A typical example is compliance to the regulations set forth in Basel II, IFRS2, MiFID3, LSF4, Tabaksblat5, and the Sarbanes-Oxley Act, just to name a few, which cover issues such as auditor independence, corporate governance, and enhanced financial disclosure.

Domain-Specific Languages (DSL): DSLs are small languages that are tailored for a particular domain. Usually, DSLs are simple because they are suited for a very narrow purpose only, and they are easy to edit and to translate. To describe a broad domain, a broad DSL can be used. To keep the smallness and simplicity of DSLs, multiple narrow DSLs should be used, which have to be combined to describe a broad domain. The goal of DSLs is to improve productivity and software quality. DSLs raise the level of abstraction to empower users with the ability to build solutions using concepts that are similar to the domain and his/her knowledge

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

Model and meta-model: a model is an abstract representation of a system's structure, function or behaviour. A meta-model defines the basic constructs that may occur in a concrete model. Meta-models and models have a class-instance relationship: each model is an instance of a meta-model.

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

Role-Based Access Control (RBAC): Access control decisions are often based on the roles individual users take on as part of an organization. A role describes a set of transactions that a user or set of users can perform within the context of an organisation. RBAC provide a means of naming and describing relationships between individuals and rights, providing a method of meeting the secure processing needs of many commercial and civilian government organisations.

Web Service Description Language (WSDL): a standard XML-based language for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

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

1.4. Abbreviations and acronyms

BPMN	Business Process Modelling Notation
BPEL	Business Process Execution Language
DSL	Domain Specific Language
EMF	Eclipse Modelling Framework
EPC	Event-Driven Process Chain
ETL	Extract-Transform-Load
LTL	Linear Temporal Logic
MDA	Model-Driven Architecture
MDSD	Model-Driven Software Development
oAW	openArchitectureWare
SOA	Service-Oriented Architecture
VbMF	View-based Modelling Framework
WSDL	Web Services Description Language
WS-BPEL	Web Services Business Process Execution Language
XPDL	XML Process Definition Language

2. Compliance Framework Architecture Overview

Many regulations, standards and internal policies act as sources of guidance for business compliance in organisations. These sources, termed compliance concerns, need to be translated into artefacts within the organization's information systems and/or business processes.

The architecture of the software framework presented in this section is designed to enable an organisation to develop and maintain compliance framework that addresses these compliance concerns. The architecture follows the model driven development paradigm. The resulting framework should be able to address both the design time and runtime issues of a business compliance infrastructure. The software framework focuses on achieving business compliance in a process-driven SOA.

Figure 1 shows the overview of the components that make up this architecture.

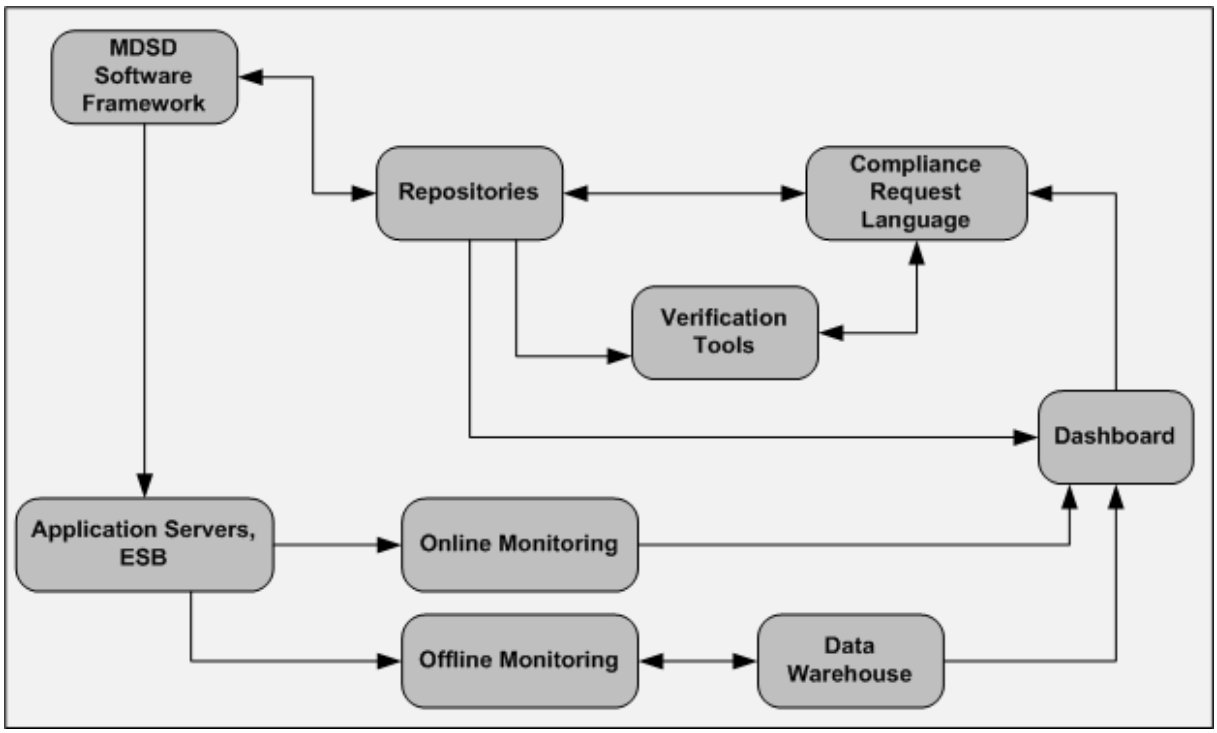

Figure 1 Overview of COMPAS Architecture

The overview of COMPAS architecture can be broken down into a number of sections;

COMPAS design time infrastructure comprises the **MDSD Software Framework**, **Repositories, Verification Tools, and Compliance Request Languages.** The MDSD Software Framework that includes View-based Modelling Framework (VbMF) presented in Section 3.2, and Domain-specific Languages (DSLs) introduced in Section 3.3 aids in the development of specialised modelling languages. In our case the specialised languages would be used to model compliance concerns and/or other supporting elements for the compliance framework. These DSLs are what the professionals in their respective domains would use to model a particular organisation's interpretation of compliance concerns. For example, the DSL components could be used to develop a modelling language for expressing only compliance concerns related to security legislation. The security professionals could then use such a DSL to develop a particular model of security that the organization is required to implement. VbMF provides tooling that enables modelling of these concerns in separate views that allow the designer an abstract view of a system specific to his/her domain. In our case, different compliance concerns can possibly be represented in different views based on the domain experts for the various categories of compliance concerns. Verification tooling allows static validation of the models and the compliance concerns whilst the Repositories provide model repositories to store models, process fragments, and compliance requirements. Compliance Request Languages are DSLs used for discovering and choosing process fragments that meet specific compliance requirements and that can be subsequently aggregated into end-to-end business processes.

COMPAS runtime infrastructure includes the **Dashboard**, and the monitors which offer the ability of both online and offline monitoring of systems and compliance under execution.

Application servers provide the platforms for deploying process engines and business services. **Enterprise Service Bus (ESB)** is the integration platform for connecting and coordinating the interaction of applications constituting the runtime infrastructure. The runtime infrastructure monitors the execution of the business processes, stores data by using data warehouses, and uses the Dashboard as an interface to users. The Dashboard then provides information about the state of the system, say status information or a compliance violation, to users.

In this section we have presented a brief overview of the various components that make up the overall compliance framework architecture. The various components are developed by different partners and in different work packages. However, while they evolve independently, these components need to be integrated to achieve the objectives set out in the project. To facilitate this integration, we now describe, in the next section, how the various components interact with each other. We describe this in the next section.

3. COMPAS Integration Architecture

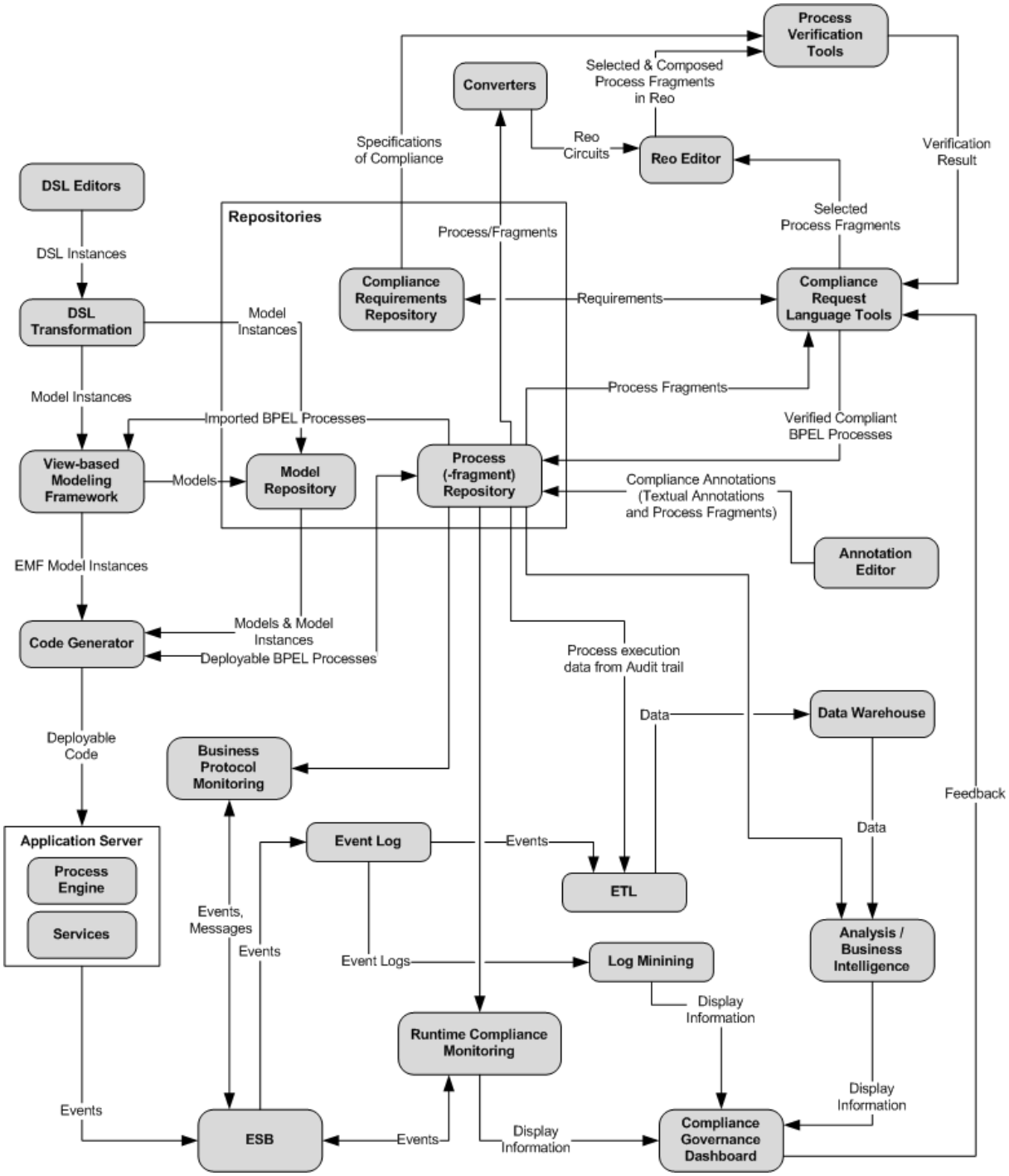

Figure 2 COMPAS architecture: interaction and integration of technologies and prototypes

In this section COMPAS integration architecture shown in Figure 1 will be discussed in detail. We introduce tools, technologies, and prototypes which are newly developed, extended or reused in COMPAS project along with the interaction and integration of them in the overall COMPAS architecture (see Figure 2). Detailed functionality, status and corresponding WP responsibility of these components are then provided in Table 1.

Every row of Table 1 describes detail of one component in the COMPAS architecture. The first column is the component's name. Next, the second column introduces functionality of these components, respectively. Each component might have one or more interactions with others. These interactions are clarified in the third column in which the corresponding component is aligned with expected inputs from its adjacent components. The forth column mentions specific technologies or tools used to realise the component as well as their status in context of COMPAS project, such as **New** (i.e., develop from scratch), **Extend** (i.e., extend existing tools or technologies), and **Use** (i.e., use existing tools or technologies without extending). The last column names the WPs involving in the component.

Component	Functionality description	Relationships to other components	Tools and technologies	Status	Work-Package
Analysis / Business Intelligence	The reasoning mechanisms and algorithms should analyse the root causes of deviations of the process constellations and fragments from the desired compliance concern targets, as well as it automatically identifies such deviations. A meta-model and DSL for compliance to security policies will be developed in this task to show-case and validate the results of this task for a technical governance concern, which will be analysed using the business process intelligence suite.	**Data Warehouse** Analysis/Business Intelligence component acts as both producer and consumer of data: it analyses data in the warehouse and stores the results back into the warehouse. **Process (-fragment) Repository** The Repository provides the Analysis/Business Intelligence component with Processes (or -fragments) that comprise relevant annotated compliance meta-data in order to support compliance-oriented analysis and interpretations.	BusinessObjects, SpagoBI	Extend	WP5
Annotation Editor	The annotation editor is a text-based editor that allows annotating artefact with other artefact. In the case of COMPAS we foresee the annotation of processes with both textual annotations and with process fragment, as well as the annotation of process fragments with textual annotations.		Text Editor	New	WP4
Application Servers	The application server is the runtime environment for components such as the process engine and the services. It is often also referred to as container.	**Code Generators** Deployable artefacts (e.g., generated and annotated processes, services, runtime configurations, etc.,) are deployed to	Apache Tomcat, Axis2 support	Extend	WP4

		process engines and application servers for execution.			
Business Protocol Monitoring	Architecture for runtime monitoring of Web Services conversations. It provides basic events as detection of compliance violation at messages level.	**Process (-fragment) Repository** The Business Process Protocol Monitoring component retrieves BPEL processes and BPEL process fragments in order to monitor business process execution. **ESB** Business Protocol Monitoring component listens to messages exchanged between Services through the ESB and publishes some events related to compliance checking into the ESB.	MS Visual Studio/ Eclipse plug-in	New	WP5
Code Generator	The Code Generator takes as inputs the modelling artefacts validated via the Model Validator, and a number of transformation templates used to produce. Then, it might perform model validations against required constraints (if any). Finally, schematic code and configurations are generated. The generated schematic code might be augmented with individual code specialized for specific business logics, particular platform features, etc.	**Model Repository** The Code Generators takes models and model-instances from the Repositories as well as templates needed for transforming model-instances to code. **Process (-fragment) Repository** The Code generator retrieves non-executable BPEL processes from the Process (-fragment) Repository and generates execution information, e.g.	openArchitectureWare	Extend	WP1

		WSDL-file, deployment descriptor etc. and stores the executable BPEL process in the Process (-fragment) Repository. **View-based Modelling Framework** View model and view instances in VbMF can be directly used by the Code Generator for generating code.			
Compliance Governance Dashboard	The Dashboards are a user friendly graphical web based visualisation of compliance information, particularly compliance violations of business process.	**Runtime Compliance Monitoring** Displays the visualised monitoring results to the user. **Log Mining** The Dashboard is fed up with analysed and interpreted results from Log Mining to display to the user. **Analysis/Business Intelligence** The analysis/business intelligence component will compute compliance indicators based on the data in the warehouse and identify (where possible) root causes of violations. Such results are displayed in the dashboard	Graphical Web UI Dashboard	New	WP5

Compliance Request Language Tools	Compliance Request Language (CRL) is a proposed formal language that can be utilised for the specification and representation of compliance requirements and introduce an initial specification of compliance language, along with its associated constructs and operators (extracted from D2.2). CRL Tools aim at supporting the user effectively making use of the CRL language.	**Process (-fragment) Repository** The Compliance Request Language Tools retrieve BPEL process fragments and BPEL processes stored in the Process (-fragment) Repository by querying the Process (-fragment) Repository employing a request language defined and specified in WP2. Moreover the Compliance Request Language Tools store verified compliant BPEL processes and process fragment compositions in the Process (-fragment) Repository. **Compliance Requirements Repository** The Compliance Requirement Repository stores and organises compliance requirements at various abstractions levels (in terms of goals, policies and rules) and allows the reusability of the compliance constraints (extracted from D2.2) **Process Verification Tools** Verified process models are stored in the Process (-fragment) Repository and can be queried by means of the Compliance Request Language Tools.	Graphical LTL tools	Extend	WP2

		Compliance Governance Dashboard Feedbacks from the Dashboard are needed by the Compliance Request Languages Tools to help the user to adjust relevant compliance requirements and specifications.			
Converters	Three converters, namely, BPMN2Reo, UML2Reo and BPEL2Reo, are Eclipse plug-ins used to convert business process models to Reo circuits, and subsequently, to constraint automata, for their formal analysis and compliance verification.	**Process (-fragment) Repositories** BPEL process fragments are retrieved from the Process (-fragment) repository are automatically converted to Reo process models for their further composition, refinement, verification and compliance analysis using Reo Editor and Process Verification Tools.	Reo, Eclipse	New	WP3
Data Warehouse	A Data warehouse is an integrated, non-volatile, historical, subject-oriented data collection, aimed at supporting decision-making processes. Particularly, in COMPAS the DW stores compliance and process related data.	**ETL** The data warehouse is the destination of the data processed by the ETL procedures. It stores transformed events in form of process, activity, and service instance facts.	Project-specific data warehouse model DBMS	New Use	WP5
DSL Editor	System requirements and compliance concerns are represented in terms of Domain-Specific Languages (DSLs). DSLs describe knowledge via a graphical or textual syntax. Therefore, DSL editors are necessary for, and often used by stakeholders to manipulate the requirements.		Eclipse-based editors, XML Editors	Extend	WP1
DSL	DSL transformations take as inputs the DSLs	**DSL Editor**	Frag, XML	Extend	WP1

Transformation	defined by using DSL editors, and then, interpret and transform them into modelling artefacts such as models, model instances and/or relevant constraints (if any).	DSL instances are passed from DSL Editors to the DSL Transformation	Parsers, Parser Generators		
ESB (Enterprise Service Bus)	The Enterprise Service Bus (ESB) is a unified communication channel between the components. Besides that the ESB provides a publish/subscribe mechanism for events that are published via this channel. In COMPAS the ESB is employed for publish subscribe mechanism for events, without implementing additional functionality.	**Application Servers** The runtime components such as the process engine executing processes, and the services describe their current status by generating and emitting events. Those events can be published via the ESB in order to inform any interested component, such as the Event Log (described below) for example. **Business Protocol Monitoring** Described above **Runtime Compliance Monitoring** Runtime Compliance Monitoring is an event subscriber of the ESB. Relevant events published by the ESB are exploited by the Runtime Compliance Monitoring for online detection of compliance violations.	ServiceMix, ActiveMQ	Use	WP2,4,5
ETL (Extract, Transform, and Load)	The ETL (extract, transform and load) processes are responsible for the extraction of the data from the Event log (possibly from Audit trail), transforming them according to	**Process (fragment) Repository** Process models will be used during ETL for the identification of executed process	Project-specific ETL procedures	New	WP5

	the DW data model.	instances out of logged events. Process fragment models can be used to check whether a specific process instance obeyed to a given fragment or not. **Event Log** The event log represents the actual data is processed by the ETL procedures. Event will be aggregated in order to reconstruct process, activity, and service instances.	Talend (or similar)	Use	
Event Log	An Event log can be seen as a set of past events typically ordered chronologically by the timestamps. Depending on the implementation the event log normally provides an interface to retrieve a certain sub-set of those events that occurred within a given interval. In the architecture of COMPAS the events are emitted from any component in form of messages, which can be delivered by the ESB that again provides publish/subscribe functionality for any component that is interested in a certain type or source of event.	**ESB** The Event Log uses the Publish-Subscribe mechanism that the ESB provides, for subscribing to, and retrieving any event that is required for further processing, such as for compliance analysis in the Data Warehouse.	DBMS	New	WP5
Log Mining	Log mining refers to the activity of extracting implicit knowledge from log repository. For instance, the actual business protocol of the process can be retrieved, allowing a better understanding of service and clients behaviour. In the architecture of COMPAS, knowledge from log mining is reported	**Event Log** The Log Mining uses subsets of events provided via the Event Log's interfaces to reason and produce relevant knowledge that are displayed to the user via the Dashboard.	Java, Matlab	New	WP5

	through the monitoring dashboard.				
Process Engine	The process engine is the component that executes processes by navigating through the steps defined in the process model, based on the current parameters of the running process instance. A process engine describes the current status of the execution of processes by generating and emitting execution events. During execution of a business process instance the engine stores additional execution data in the audit trail e.g., incoming purchase order, and emits events to the ESB, which is used for reliable messaging and Publish-Subscribe mechanism for event messages		Apache ODE	Use	WP1,5
Process Verification Tools	Process Verification tools include a Reo animation plug-in and a Vereofy model checker. Reo animation plug-in is a tool that generates flash animated simulations of formal business process models. The plug-in depicts the process that was previously shown in the Reo editor in the animation view. The parts of the process highlighted red represent synchronous data flow. Tokens move along these synchronous regions. On the left side there is a list of possible animations for this connector and the attached writers and readers. Reo validation plug-in is a tool that performs model checking over coordination models	**Reo Editor** Process Verification Tools take as input format constraint automata automatically generated from Reo process models. **Compliance Requirement Repository** Compliance requirements are expressed as logic formulas and used as input for the model checker.	Eclipse plug-in	Extend	WP3

	represented as constraint automata. This model checker uses a symbolic model and LTL logic as property specification formats.				
Reo Editor	The Reo editor is an Eclipse plug-in that enables business process modelling by simple drawing operations and serves as a bridge to a number of other tools that can be either invoked from the context menus or directly interact with it. Formal business process models are stored using an XML format and can be further verified and transformed to service compositions by wiring appropriate web services.	**Converters** Reo process models can be automatically obtained from BPMN/UML diagrams (green field scenario) or BPEL process fragments with the help of corresponding converters.	Reo, Eclipse	Extend	WP3
Repositories	The Repositories provides means for registering, persisting, and versioning modelling artefacts in order to enhance reusability and collaborative development. There are three main types of repositories including Compliance Requirement Repository, Model Repository and Process (-fragment) Repositories. • Compliance Requirement Repository • Process (-fragment) Repository. • Model Repository	**DSL Transformation** Model instances produced by the DSL Transformation are stored in the Model Repository for later use. **Compliance Request Language Tools** The Compliance Request Language Tools retrieve BPEL process fragments and BPEL processes by querying the Process (-fragment) Repository employing a request language defined and specified in WP2. Moreover the Compliance Request Language Tools store verified compliant BPEL processes and process fragment	DBMS as backend	New New New	WP2 WP4 WP1

		compositions in the Process (-fragment) Repository. **Annotation Editor** The Annotation Editor is employed for annotating BPEL processes with textual annotation as well as process fragments for defining and specifying compliance constraints. Besides the BPEL processes and BPEL process fragments are annotated with meta data, e.g. information about application domain. For details see [D4.1] **Runtime Compliance Monitoring** The Runtime Compliance Monitoring component retrieves BPEL processes and BPEL process fragments in order to do near-real time monitoring of business process execution. **Code Generator** Schematic process code generated from the Code Generator can also be stored in the Process (-fragment) Repository. Conversely, processes and process fragments in the repository can be queried			

		and re-used within the Code Generator.			
Runtime Compliance Monitoring	Software architecture for online detection of compliance violations. It's based on complex event processing concepts and provides immediate notification of detected violations.	**Process (-fragment) Repository** The Runtime Compliance Monitoring component retrieves BPEL processes and BPEL process fragments stored in the Process (-fragment) Repository in order to do near-real time monitoring of business process execution. **ESB** Described above	CEP engine, Eclipse	New	WP5
View-based Modelling Framework	View-based Modelling Framework acts as a modelling foundation for representing different process concerns by exploiting the concept of architectural views. Process concerns, such as the control-flow, service invocations, data handling, etc., are modelling artefacts. These artefacts might be bound to some modelling constraints, or be associated with meta-data of some compliance concerns.	**DSL Transformation** The DSL transformation parses the DSL instances and produces model instances that can be manipulated by the View-based Modelling Framework. **Process (-fragment) Repository** The View-based Modelling Framework imports BPEL processes from the Process (-fragment) Repository, transforms them into EMF-models and stores these models in the Model Repository.	EMF, Frag	Extend	WP1

Table 1 Mapping of COMPAS components into prototypes, tools and/or technologies

3.1. Model-driven Integration Architecture

After describing the dependencies of the various components that compose the compliance framework, it needs to be emphasized that one of the objectives of the project is to develop a framework that "enables companies to rapidly develop and then stably evolve and maintain a business compliance framework".

To do this, we leverage the MDSD paradigm that addresses some of the challenges (like the maintenance, evolution and reuse of systems and system components) experienced during the design time of compliance solutions. In this section, therefore, we describe the structure of the MDSD environment that we intend to use for this purpose.

As mentioned in the overview, the design time infrastructure consists of MDSD Software Framework, Repositories, Verification Tools, and Compliance Request Languages. In this section, we present a model-driven tool-chain (see Figure 3) which is the fine detail of the **MDSD Software Framework** component shown in COMPAS overall architecture (see Figure 1 and Figure 2). This component provides basic concepts and mechanisms for integrations of COMPAS prototypes, tools, and technologies.

At design time, stakeholders supporting by appropriate tools and editors develop view models for business processes and specify relevant compliance requirements that those processes comply with. These compliance requirements are described in terms of DSL concrete syntax. DSL parsers take the DSL instances representing compliance requirements as inputs and transform them into model instances. Transformation mechanisms implemented in DSL parsers are defined according to corresponding models, parser specifications, and model - mapping specifications. After that, the Model Validator performs constraint checking on each model instance against its model and associated constraints. Model instances qualifying the constraint checking are candidates for transforming into EMF models by the Model Transformation plug-in. The code generator component which is an extension of openArchitectureWare model transformation [OAW] consumes the EMF models and generates system code and configurations being deployed in process engines, application servers, and other relevant COMPAS components, for instance, the governance framework, the dash board. The transformation templates define schematic code and configuration being generated by the code generator. Moreover, necessary individual code that implements specific business logic might be injected into schematic code in order to fully accomplish particular desired functionalities.

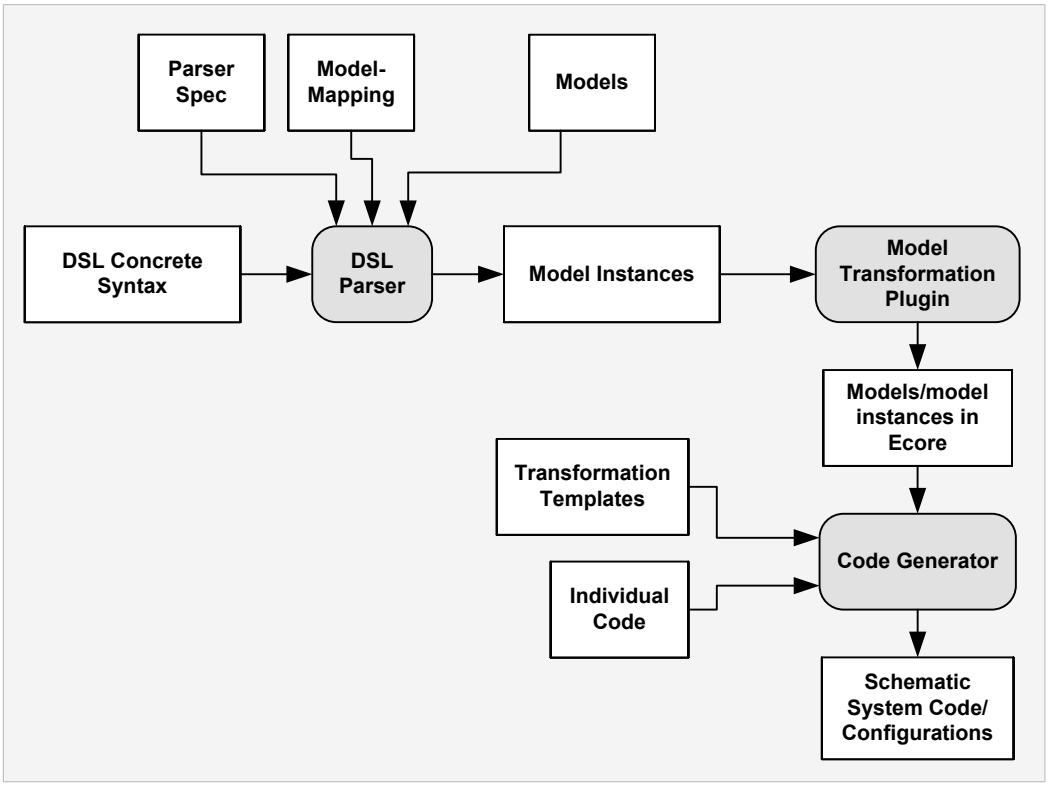

Figure 3 Foundation of integration architecture: a view-based, model-driven tool-chain

The subsequent sections gradually introduce important concept constituting the tool-chain. Section 3.2 provides overview of the view-based framework which is the basis for COMPAS model-driven architecture. Section 3.3 is dedicated to domain-specific languages that can be utilised as a vehicle for representing compliance concerns.

3.2. View-based Modelling Framework (VbMF)

In a process-driven, service-oriented architecture (SOA), business functionality is accomplished by executing business processes invoking various services. A typical business process includes a number of activities and a control flow. Each activity corresponds to a communication task (e.g., it invokes other services or processes) or a data processing task. The control flow describes how these activities are orchestrated. A process is typically represented either in an executable language, such as BPEL [BPEL] or XPDL [XPDL], or in a high-level modelling language such as BPMN [BPMN], EPC [EPC], or UML Activity Diagrams [UML].

Business and domain analysts who understand business and domain concepts best design processes in high abstraction languages, such as BPMN, EPC, or UML Activity Diagram. These designs are handed over to IT experts who will implement the processes using executable languages, such as BPEL, and deploy these processes on process engines. Bridging the gap between process design and implementation is challenging because of the complexity of the process descriptions, the divergence of process modelling languages in terms of syntax and semantics, and the discordance of levels of abstraction and granularity [HZD07, HZD08a, HZD08b]. In [HZD07, HZD08a, HZD08b], we proposed a view-based model-driven framework for process-driven SOAs to address these problems. The view-based approach serves as the basis for COMPAS model-driven integration architecture. In

subsequent sections, we introduce basic concepts of the view-based modelling framework with relevant extensions for modelling of business compliance, and describe useful modelling mechanisms such as view extension and code generation that support collaboration and integration between WP1 and other WPs.

3.2.1. Overview of the View-based Modelling Framework

In this section, we briefly introduce the View-based Modelling Framework [HZD07, HZD08a, HZD08b]. Figure 4 shows VbMF with proposed extensions for modelling of business compliance. From now on, the term "VbMF modelling framework", unless specifically stated otherwise, is used to mention the combination of VbMF and the proposed modelling extensions as well.

The modelling framework consists of modelling artefacts such as a meta-model, view models, and view instances (see Figure 4). In VbMF, an architectural view (or view for short) is a representation of a process from the perspective of related concerns. Each view instance comprises many relevant elements and relationships among these elements. The appearance of view elements and their relationships are precisely specified in a view model that the view must conform to. A view model, in turn, conforms to a MOF-compliant meta-model [MOF] derived from Eclipse Modelling Framework meta-model [EMF]. VbMF view models are devised on top of that meta-model.

On the left-hand side, the modelling framework provides basic view models for describing process-driven systems. As stated in [HZD07] three view models: Flow, Collaboration, and Information view model, represent the basic concerns of a business process. Other concerns, for instance, transaction, human integration, event handling, etc., are also developed and plugged into VbMF accordingly thanks to its extensibility. For the sake of simplicity and concentration on compliance modelling, other view models are not presented in Figure 4. On the right-hand side, VbMF is going to be leveraged for modelling of compliance concerns. These compliance concerns will derived from the basic facilities provided by the common meta-model. The aggregation of desired compliance concerns to complement a certain process in order to make it compliant is so-called the Compliance Meta-data Model. Being annotated with a relevant Compliance Meta-data Model, the process models are readied for generating schematic process code as well as configurations needed to deploy and monitor the execution of the business process.

The goal of the modelling framework, as mentioned in [DoW], is to provide concepts and solutions for supporting all kinds of business compliance throughout a SOA. Due to the limitation of project duration, COMPAS will address a subset of the compliance [DoW] including following concerns:

- Quality of Service (QoS) policies (contributed by WP1)
- Security policies (contributed by WP5)
- Intellectual property and licenses (contributed by WP5)
- Regulatory or legislative provisions (contributed by WP2)

Regarding particular compliance requirements, the view models represented business processes will be annotated or associated with corresponding compliance meta-data. Our modelling framework is not bound to four compliance concerns mentioned above. It is possible to extend the framework into other compliance concerns using the extension mechanisms provided in [HZD07].

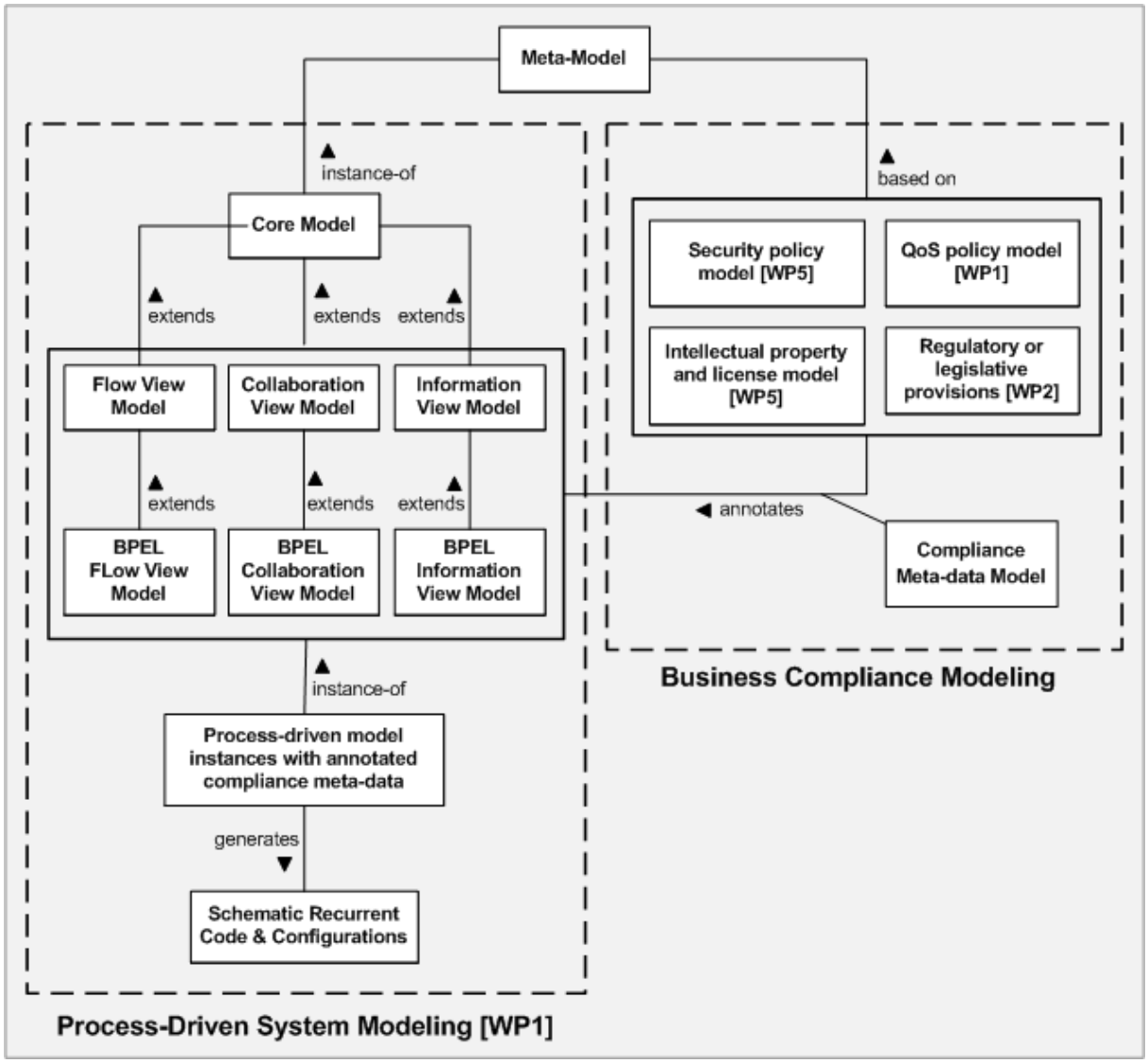

Figure 4 The View-based Modelling Framework with extended facilities for modelling of compliance concerns

3.2.2. View-based Modelling Framework Architecture

In VbMF, we categorise distinct components in which the modelling artefacts are manipulated (see Figure 5) [HZD07]:

- *View Model Editors* are based on view models. Using these editors, a new view model might be developed from scratch by deriving the Core model. Moreover, an existing view model might also be extended with some additional features to form a new view model.
- *View Model Instance Editors* can be (semi)-automatically generated from VbMF view models. These editors support stakeholders in creating new view instances or editing existing instances.
- *View Integrators* aid the stakeholders in combining view instances to produce a richer view, or a thorough view of a certain business process.

- *Code Generators* generate executable code from one or many view instances. Before generating outputs, the code generators will validate the conformity of the input views against corresponding view models.
- *View Interpreters* are used to extract relevant views from existing legacy business process code.

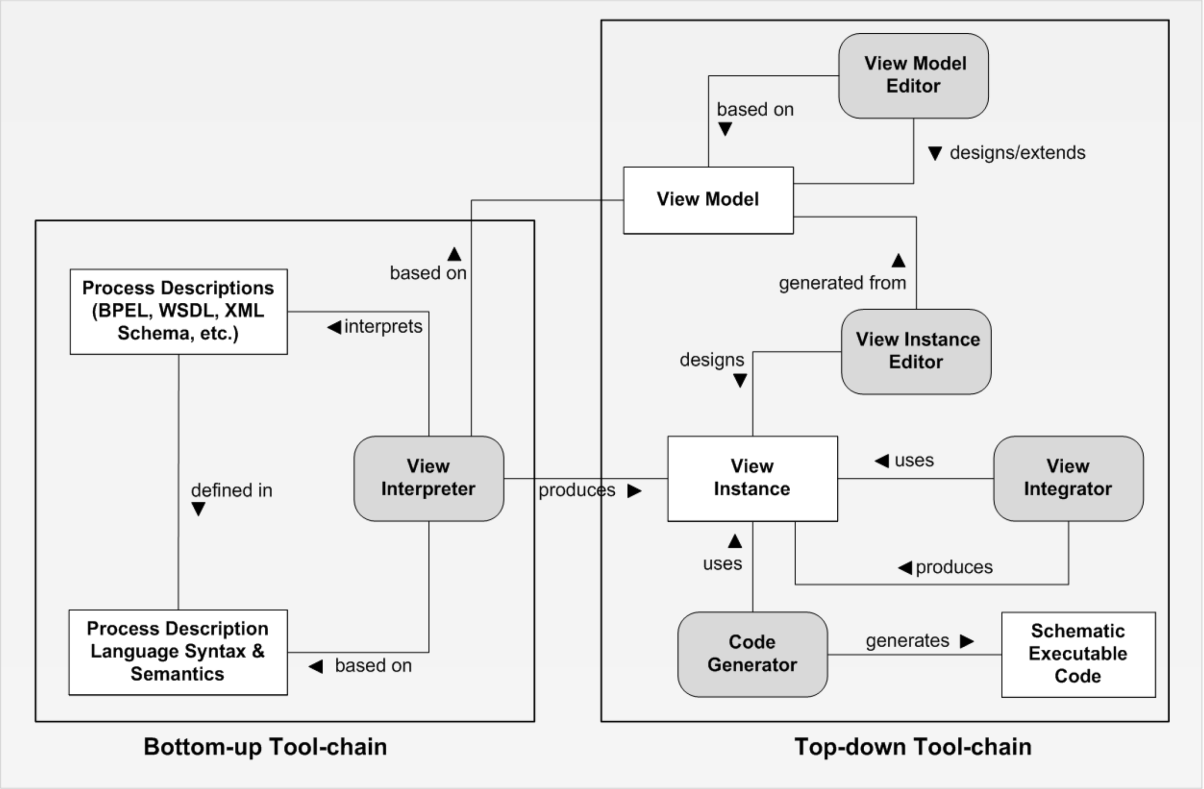

Figure 5 Top-down and bottom-up tool-chains in View-based Modelling Framework

In VbMF top-down tool-chain high level view instances are designed first. Then, these instances are refined down to their lower level counterparts which are technical-specific view instances. The code generator uses the technical-specific view instance in order to produce schematic process code and/or necessary configurations. In the course of the bottom-up tool-chain, the view interpreters take as input legacy process descriptions and produce high level or low level view instances that can be re-used later in the top-down tool-chain.

3.2.3. Supporting MDSD mechanisms in VbMF

3.2.3.i. Extension mechanisms [HZD07]

During the process development lifecycle, various stakeholders take part in with different needs and responsibility. For instance, the business experts - who are familiar with business concepts and methods - sketch blueprint designs of the business process functionality using abstract and high level languages such as flow-charts, BPMN diagrams, or UML activity diagrams. Based on these designs, the IT experts implement the business processes using executable languages such as BPEL, XPDL, etc. Hence, these stakeholders work at different levels of abstraction. According to the specific requirements on the granularity of the view

models, we can gradually refine these models toward more concrete, platform- or technology-specific views using the extension mechanisms [HZD07].

A view refinement is performed by, firstly, choosing adequate extension points, and consequently, applying extension methods to create the resulting view. An extension point of a certain view is a view's element which is enhanced in another view by adding additional features (e.g., new element attributes, or new relationships with other elements) to form a new element in the corresponding view. Extension methods are modelling relationships such as generalisation, extend, etc., that we can use to establish and maintain the relationships between an existing view and its extension. For instance, the Flow View, Collaboration View, and Information View models are mostly extensions of the Core model using the generalization relationship. In the same way, more specific view models for other technologies can be derived. In addition, other business process concerns such as transactions, event handling, and so on, can be formalised by new adequate view models derived from the basic view model using the same approach as used above [HZD07].

3.2.3.ii. Integration mechanisms

In VbMF, the Flow View model - as the most important concern in process-driven SOA - is often used as the central model. View models can be integrated via integration points to provide a richer view or a thorough view of the business process. In [HZD07], named-based matching mechanism is used for integrating view models. This mechanism is effectively used at model level because from a modeller's point of view, it makes sense and is reasonable to give the same name to the modelling entities that pose the same functionality and semantics. However, other integration approaches such as those using class hierarchical structures or ontology-based structures are applicable in the view-based modelling framework [HZD07].

3.2.3.iii. Reverse engineering of legacy process code [HZD08a, HZD08b]

In the context of process-driven SOAs, many existing systems have built up an enormous repository of existing process code in executable languages, for instance, BPEL. These languages are rather technology-specific and therefore the abstract representations are not explicitly available at the code level. As a result, the process models become too complex for stakeholders to understand and maintain, to integrate, to cooperate with other processes, or to re-use process models from existing modelling tools.

VbMF can potentially resolve these issues. However, for budgetary reasons, developing the view models, required in our approach, from scratch is a costly option. The alternative is an (automated) re-engineering approach comprising two activities: reverse-engineering for building more appropriate and relevant representations of the legacy code; forward-engineering for manipulating the process models and for re-generating certain parts of the process code. During the reverse engineering process, high-level, abstract and low-level, technology-specific views on the process models are recovered from the existing code. This way, the reverse engineering approach helps stakeholders to get involved in process re-development and maintenance at different abstraction levels. Reverse engineering of business processes not only helps to adapt process models to stakeholder needs but also offers the ability to integrate various process models to enhance the interoperability of process models [HZD08a, HZD08b] (see Figure 6). The view interpreters play a central role in the bottom-up tool-chain in VbMF (see Figure 5).

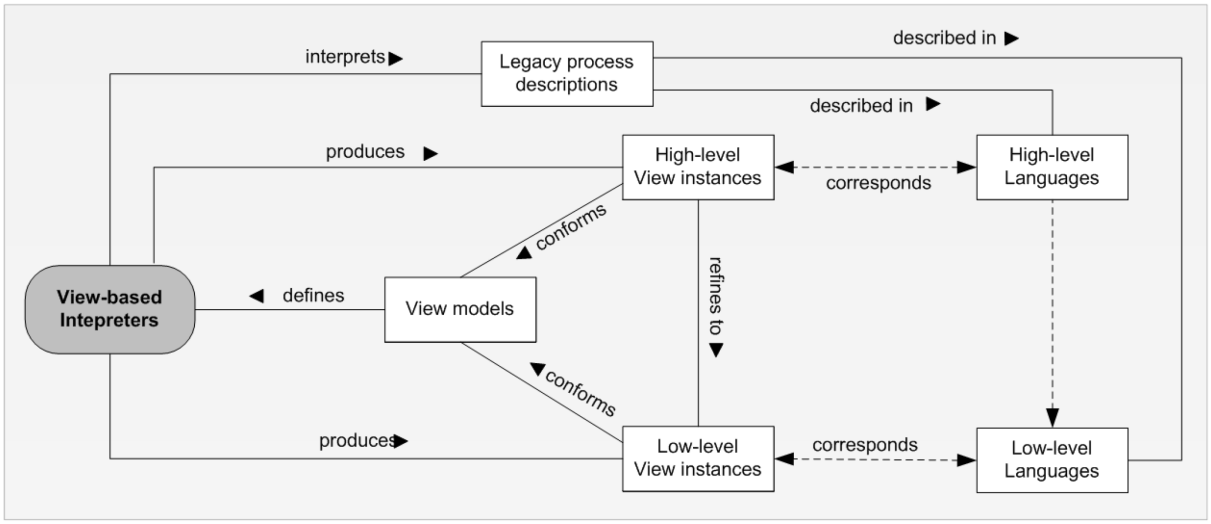

Figure 6 Reverse engineering of legacy code in VbMF

3.2.3.iv. Model-to-code transformations [HZD07]

There are two basic types of model transformations: model-to-model and model-to-code. A model-to-model transformation maps a model conforming to a given meta-model to another kind of model conforming to another meta-model. Model-to-code, so-called code generation, produces executable code from a certain model. In VbMF, the model transformation is model-to-code that takes as input one or many views and generates code in executable languages, for instance, BPEL and WSDL [HZD07]. VbMF utilized the combination of template and meta-model technique realized in the openArchitectureWare framework [OAW] to implement the model transformations [HZD07] (see Figure 7). Nonetheless, other of above-mentioned techniques could be utilized in VbMF with reasonable efforts as well.

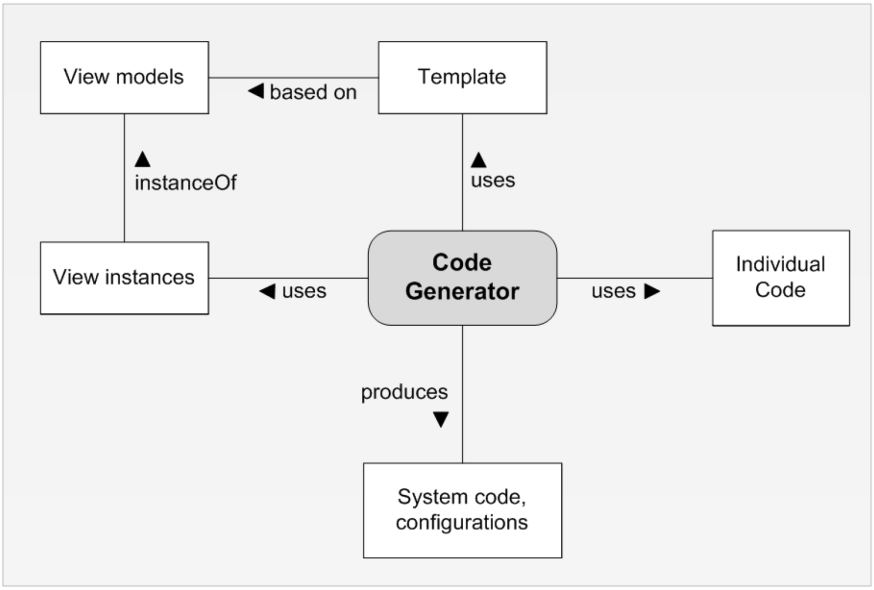

Figure 7 VbMF code generator (adapted from [HZD07])

3.3. Domain Specific Languages for Compliance Concerns

This section will demonstrate how to specify a Domain-Specific Language (DSL) for the defined compliance models. First, a general introduction to DSLs is given. Second, the

development of DSLs based on the Model-Driven Software Development (MDSD) paradigm is described. The section concludes with an example of a DSL which is used to describe Quality-of-Service (QoS) compliance concerns.

3.3.1. What are Domain-Specific Languages?

Domain-Specific Languages (DSLs) are small languages that are tailored for a particular domain. In the area of DSLs, the term domain is related to a certain domain of the real world, e.g., the banking domain [PM06]. Usually, DSLs are simple because they are suited for a very narrow purpose only, and they are easy to edit and to translate. To describe a broad domain, a broad DSL can be used. To keep the smallness and simplicity of DSLs, multiple narrow DSLs should be used, which have to be combined to describe a broad domain. Describing a domain with multiple DSLs is also called Language Oriented Programming [Fow05].

The goal of DSLs is to improve productivity and software quality. In contrast to General Purpose Languages (GPL), such as Java or C#, a DSL serves to accurately describe one domain of knowledge. A DSL concentrates on the efforts of the stakeholders to describe the problems of the domain, while complexity, design, and implementation decisions and details are hidden.

DSLs raise the level of abstraction to empower users with the ability to build solutions using concepts that are similar to the domain and his/her knowledge [SSL+07]. All different DSL users can specify the solutions with a familiar vocabulary of the problem domain [PM06].

One trade-off of DSLs lies in the high initial investment phase required for designing and developing. Furthermore, inflexibility with regard to the target platform is given. In most cases, the code generators only support a particular target platform. Due to changing technologies and platforms, code generators need to be maintained permanently.

But, some few advantages of using and/or developing DSLs are as follows:

- A clear view of the problem domain is given.
- Only valid relations between the domain concepts exist.
- Due to the separation between business and technical level, multiple levels of abstractions exist.
- The productivity and the software quality can be improved because they are easier to maintain and the generated code does not contain bugs.
- The generated code can be tailored for the particular technology.
- Technical aspects are not shown to business experts.

3.3.2. DSLs based on MDSD

Because MDSD provides high levels of abstractions and platform-independency, a very common development approach for DSLs is MDSD. Our approach of MDSD-based DSLs, also proposed in [LJJ07], is depicted in Figure 8. A DSL consists of an abstract and concrete syntax. The abstract syntax, which is based on a meta-model, defines the elements of the domain and their relationships without considering their notations. The meta-model defines how the domain elements and their relations must be described [VS06]. The concrete syntax describes the representation of the domain elements and their relationships in a human readable form. Abstract and concrete syntax are used that DSL users can define model instances which represent a particular problem of the domain. Transformations, which are

defined on the model, transform the user-defined model instances into schematic recurring code.

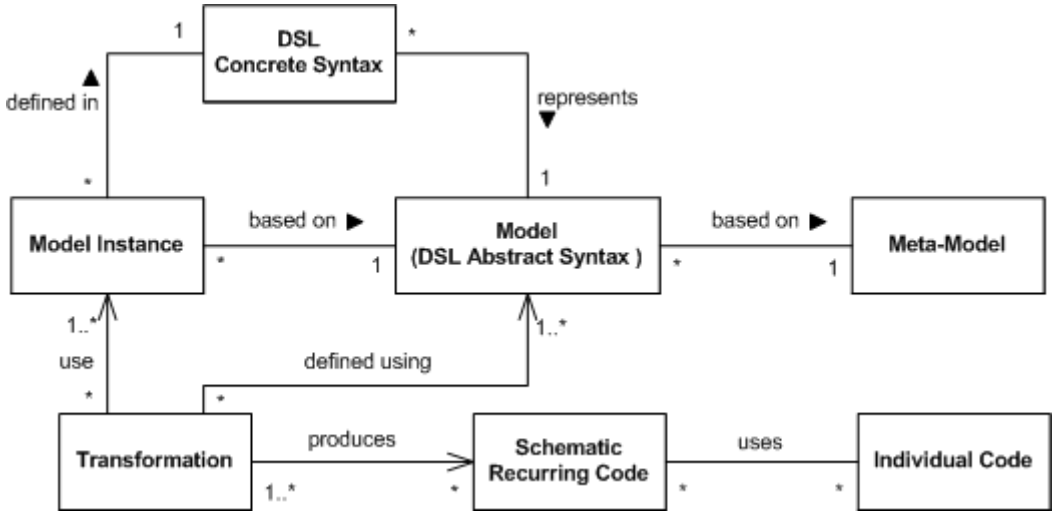

Figure 8 MDSD based DSL – Relevant Concepts

Our approach of developing DSLs aims to dividing the DSL into multiple DSLs, as demonstrated in Figure 9. Domain experts can work in a language, from now on called high-level language, where the syntax is close or equal to the domain terminology. On the other hand, technical experts can express the additional technical aspects with a language, from now on called low-level language, where the syntax is close or equal to terminology of the used technology. In DSLs, the syntax of the high- and low-level languages is based on language models. Low-level language models are extensions to high-level language models. In this way, technical experts are able to add the additional needed technical aspects. DSLs are then used to define instances of the high- and low-level language models. Each instance contains the concrete solution of a certain problem.

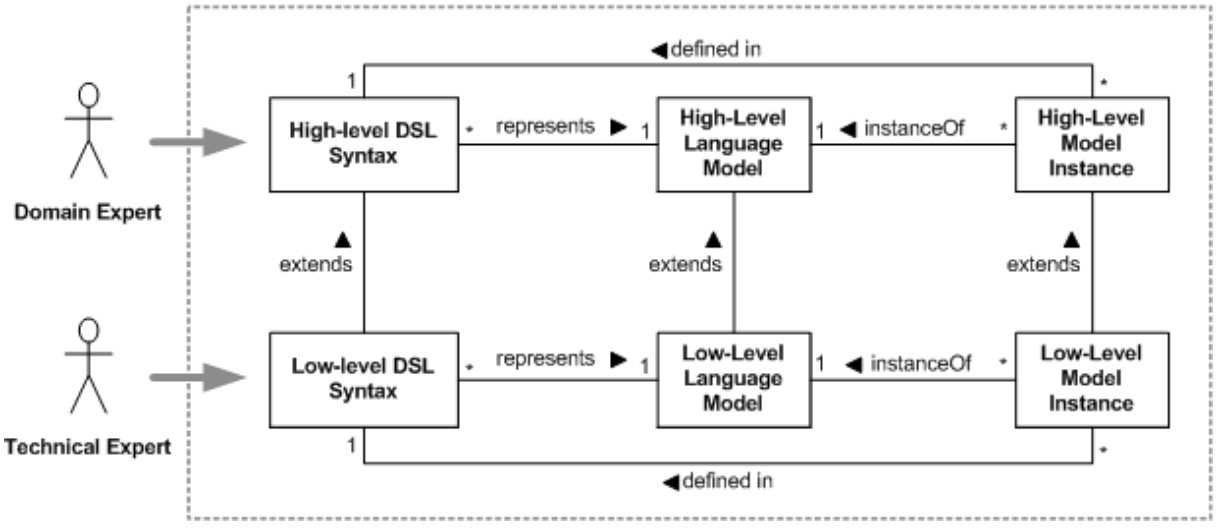

Figure 9 High- and Low-Level DSLs

Due to the multiple levels of abstractions, business or domain experts do not need to know any details about the underlying technology. For instance, the abstraction level for business or domain experts of the banking domain can provide terminologies and notations like account,

customer, withdrawal, or stock order. On the other hand, the abstraction level for technical experts can provide technical terminologies and notations, e.g., database connection, Web service, operation, or parameter. Hence, technical experts can express the additional needed technical aspects of the needed solution with a familiar terminology [OZD08].

The development process of DSLs brings many design decisions and tradeoffs. An example is demonstrated by developing a DSL for XML. A design decision regards the notations of the concrete syntax of the DSL. Should the notations be close to XML or should they be named based on names given by domain experts? On the one hand, the XML syntax can be parsed easier, but, the customised syntax is actually easier to understand for domain experts. This example shows that design decisions bring tradeoffs [Fow05]. Hence, the responsible developers of the DSL have to handle with those design decisions and tradeoffs.

The problem of companies - or rather their expert developers - is how to come up with a suitable DSL. If the DSL reflects the code the developers usually write, the possibility for productivity is limit. DSLs should focus on restricted and narrow domains the companies have been working on already. Hence, the level of abstractions gets enhanced, and it is easier to create code generators to automate development [Tol08].

3.3.3. A DSL for Specifying Locative Compliance Concerns

This section describes the specification of a model on which a DSL is based on, and how the DSL can be used. Especially, we concentrate on how locative compliance concerns can be defined for business processes. A more detailed introduction to locative compliance concerns is given in Section 4.3. The following DSL is defined and used in Frag [FRAG].

Listing 1 describes how the model of the DSL can be defined. Frag provides constructs for specifying the classes and their attributes of the model. Also, constructs for defining relations (i.e. associations or compositions) are provided. In our case, we define some classes, i.e., `ExecutionCompliance`, `Process`, `ProcessActivity`, and `ExecutionLocation`, where the `ExecutionLocation` class has a number of attributes. Then, two associations are defined. The first one is an association between the classes `ExecutionCompliance` and `ExecutionLocation`. The second on is between the classes `ExecutionCompliance` and `Process`.

```
## first, create the classes of the model
## and define their attributes
FMF::Class create ExecutionCompliance
FMF::Class create Process
FMF::Class create ProcessActivity
FMF::Class create ExecutionLocation -attributes {
      domain          String
      IP              String
      host            String
      country         String
      geocordinates   String
      priority        int
}

## second, define the relations between the classes
FMF::Association create ExecutionLocationCompliance -ends {
      {ExecutionLocation -roleName location -multiplicity 1}
      {ExecutionCompliance -roleName compliance -multiplicity 1}
}

FMF::Association create ExecutionComplianceProcess -ends {
      {ExecutionCompliance -roleName compliance -multiplicity *}
      {Process -roleName process -multiplicity *}
}
```

Listing 1 Definition of the DSL Model for Locative Compliance Concerns

After the definition of the DSL model, the DSL users can define concrete problems of the domain based on the DSL model. The following code, Listing 2, gives an example of how DSL users can assign locative compliance concerns to a process.

```
ExecutionCompliance create aLocationCompliance
      -location [ExecutionLocation create aLocation
                  -set domain "infosys.tuwien.ac.at"]
      -process [Process create aProcess]
```

Listing 2 Definition of a Locative Compliance Concerns

The DSL users can define concrete problems of their domain within the DSL. The code example above defines an *ExecutionCompliance* that specifies the *location*, and assigns it to a *Process*.

3.3.4. A Sample DSL – Quality-of-Service (QoS) DSL

This section gives a more detailed example of how MDSD-based DSLs can be divided into high- and low-level DSLs which can be used by business and technical experts, respectively. While the next deliverable of WP1, Deliverable 1.2, will define concrete syntax and semantics

for DSLs in more detail, we do not focus on these aspects within this deliverable. Instead, we complete the introduction of high- and low-level DSLs with an example DSL.

The purpose of the following DSL is to specify Quality-of-Service (QoS) of Web services, as well as actions which should be performed if Service Level Agreements (SLAs) get violated. Business experts should be able to specify which QoS values have to be measured on a particular Web service to fulfil the agreed SLAs, as well as actions or events which should be performed if a certain SLA is violated, e.g., if the response is longer than 2 minutes, then send an e-mail to the administrator of the service provider. On the other hand, technical experts need something to specify how QoS values are measured on a particular technology, as well as how the defined actions are executed. Only the collaboration between business and technical DSLs results in a running system.

To achieve the aims of specifying QoS of Web services for business and technical experts, we provide two DSLs. The first one, the high-level language, is tailored for business experts. It provides constructs and expressions that are named similar to the terminology of the particular domain the DSL was designed for. The second one, the low-level language, is tailored for technical experts. The low-level language is an extension of the high-level one, because it enriches the high-level language with technical concerns, e.g., how to measure the response time on a particular Web service engine. Similar to high-level DSLs, the constructs and expressions of the low-level DSL are named similar or equivalent to the appropriate technology.

In the following we will describe the models on which the high- and low-level DSLs are based on, and how the high-level models get extended by the low-level ones.

- **High-Level Model**

  The requirements for the high-level DSLs can be formulated as follows: SLAs are associated with Web services, as well as with actions. SLAs depend on measured QoS values, where, for the time being, the main attention lies on performance QoS values, e.g., response time, wrapping time [RPD06].

  Figure 10 depicts the model of the high-level QoS DSL. `Services` are associated with `QoSMeasurements`. For the time being, we provide classes for measuring `Performance` and `Dependability` QoS measurements, as described in [RPD06]. Each `QoSMeasurement` can have `Service-Level-Agreements` (SLAs) which are in relation with different `Actions` that should be raised if an SLA gets violated. E.g. if the `ResponseTime` of a `Service` is longer than 2 minutes, a `Mail` should be send to the service provider.

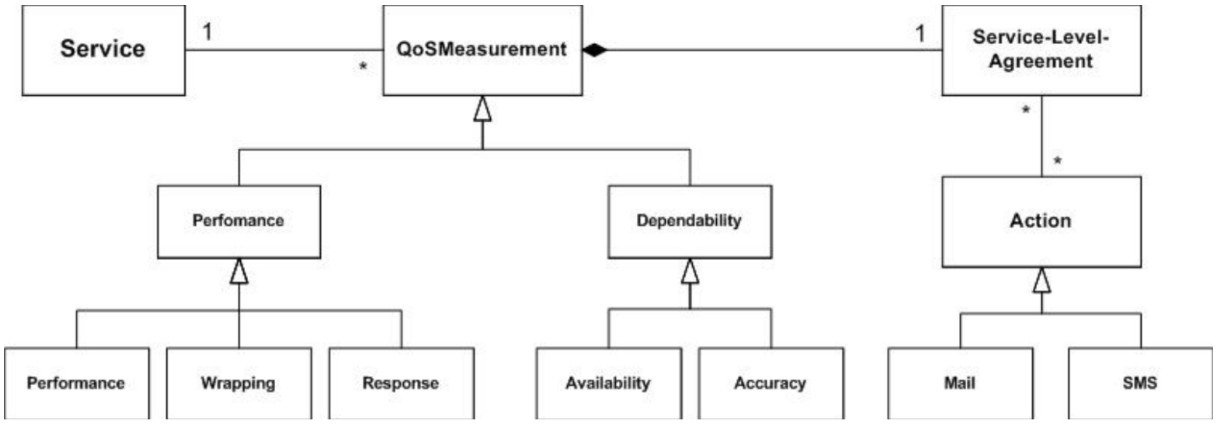

Figure 10 High-Level Model of the QoS DSL

The high-level model is extended by the following described low-level model which contains all necessary constructs for specifying the technological aspects to get a running system.

- **Low-Level Model**

  The expressions of the low-level DSLs depend on the technology on which the DSL is based on. We decided to use the open-source Apache CXF Web service framework [CXF]. The requirements can be modelled as follows: The communication between clients and services is based on message-flows. Each message-flow consists of a number of phases, where each phases can contain handlers for measuring QoS values. E.g. the handler for measuring the response time is associated to two phases of the message flow on the client side.

  Figure 11 depicts the low-level model of the QoS DSL, and how the low-level model extends the high-level model. The `Service` class at the low-level model extends the `Service` class at the high-level model. Services are enriched with `Operations` which have a particular number of `Parameters`. For measuring QoS values of services, e.g., the response time, `QoSHandlers` are associated to services. Again, QoS handlers are associated with different QoS measurements of the high-level model. In our case, the QoS handler class extends the `Response` class of the high-level model. In the Apache CXF framework, QoS handlers are associated to `Phases` where each phase corresponds to a certain `MessageFlow`. Now, the technical experts can specify in which `Phases` of which `MessageFlows` the `Response` time of a `Service` is measured when the underlying technology is the Apache CXF framework.

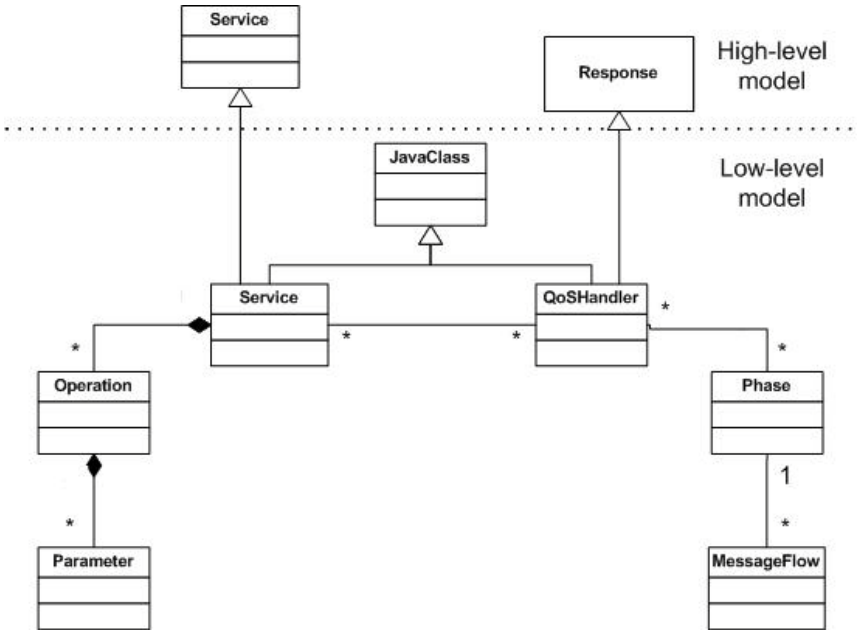

Figure 11 Low-Level Model of the QoS DSL

Next, we want to demonstrate how the models and the DSLs are combined, so that business and technical experts can use the appropriate tailored high- and low-level DSLs. The following demonstrated DSLs were developed and used within Frag [FRAG]. While business experts may directly use concrete textual DSLs for expressing compliance concerns, also graphical DSL tools are possible for representing and populating abstract DSLs.

- **The QoS DSL for Business Experts**

```
## define a service and add some measurements to it
Service create QoSService
-measure [ResponseTime create QoSResponseTime
          -assert [SLA create ResponseAssertion
                        -set predicate "LONGER THAN"
                        -set value "10"
                        -set unit "SECONDS"
                        -set actions [Mail create SendMailToProvider
                                      -set mailto "admin@provider"]]]
```

Listing 3 The High-Level QoS DSL for the Domain Experts

First, the business user has to create a Web service and specify which QoS values should be measured. An `Assertion` is assigned to the `ResponseTime` where the SLAs are specified. The idea of specifying a predicate, a value, and a unit for QoS assertions is taken from [REM+07]. Actions, which specify what should happen if a violation against the SLAs occurred, are assigned to `Assertions`.

In our example, a service, `QoSService`, is created where the `ResponseTime` will be measured. An SLA, `ResponseAssertion`, is defined which occurs if the `ResponseTime` is `greater than 10 seconds`. If so, a `Mail` should be sent to the service provider which has the e-mail address `admin@provider`.

Technical experts have to enrich the, by the domain experts specified aspects, with technical aspects to become a running system. The next paragraph shows how technical aspects can to be described by using the low-level DSL.

- **The QoS DSL for Technical Experts**

  First, the technical expert, in our case an Apache CXF [CXF] expert, specifies how the message flow between the service client and the service is done. Client and service have in- and out-flows, where in-flows are responsible for the incoming of a message and out-flows are responsible for outgoing messages. Each flow consists of phases. After specifying the phases, the technical expert defines where each QoS value has to be measured.

```
## define message flows of client
ClientFlow create ClientInFlow -superclasses ClientFlow
ClientFlow create ClientOutFlow -superclasses ClientFlow

## define phases of the message flows
OutPhase create OutSetup
OutPhase create OutSetupEnding

## assign phases to message flows
ClientOutFlow phases {OutSetup OutSetupEnding}

## define in which phases the response time is measured
ResponseTime measuredInFlows {ClientOutFlow}
ResponseTime measuredBetweenPhases {OutSetup OutSetupEnding}
```

Listing 4 The Low-Level QoS DSL for the Technical Experts

In our example, the in- and out-flows of the service client, `ClientInFlow` and `ClientOutFlow`, are specified. Then, the phases of the out-flow, `OutSetup` and `OutSetupEnding`, are defined. In the end, the technical experts specify the flows and phases, between which the `ResponseTime` is measured.

3.3.5. Tools for DSL-Development

Nowadays, many tools exist for developing DSLs, whether based on MDSD or not. Using Frag [FRAG] for developing and using DSLs, which is described above, is not obligatory although we provide an automated model-driven generation of Frag code that is ready to use for internal DSLs. The Frag example is provided to give better insides for a better understanding to the reader. In order to bind external, concrete DSLs to the abstract DSLs we provide an example with a parser implementation that can be adapted for individual DSLs.

Other tools for developing DSLs are:

- Microsoft DSL Tools [MSDSL]
- Microsoft Oslo [OSLO]
- XText which is provided by openArchitectureWare [OAW]

4. Compliance Modelling

After having introduced MDSD, the View-based Modelling Framework and DSL support, we will now start with a general discussion on compliance concerns and identify and extract various concepts. In Section 4.1 we will then show how to integrate these concepts with the VbMF and give an example on how to model a business process with some compliance aspects.

4.1. Compliance Views

Section 3.1 of [D2.1] gives an overview of various compliance concerns such as locative, monitoring, quality or security concerns and their relation to several compliance legislations such as Basel II [BASELII] or Sarbanes-Oxley Act. [SOX].

In order to express the compliance of a certain process or process activity we propose two options: An existing process or process activity can be annotated to be compliant to a certain compliance rule of a specific regulation as probably implemented by a standard framework. This allows for reuse of legacy processes that need to be marked as compliant. Such processes would have to be validated manually, thus no validation at runtime may occur using this approach. Also the compliance for a process or process activity can be modelled. In this case for each of the compliance concerns an appropriate precisely specified model has to be defined using modelling techniques. Besides the modelling of compliance concerns also Object Constraint Language (OCL) [OCL] constraints can help to restrict the models in order to become compliant. Therefore all models may be extended by additional compliance OCL constraints.

[D2.1] distinguishes between basic and advanced compliance concerns. For each of the basic compliance concerns we will come up with a starting example. There, we will consider a scenario with a specific compliance requirement that addresses the corresponding compliance concern in a simplistic way. We will then illustrate a pragmatic approach how to address this requirement in regard to compliance modelling using VbMF. A resulting model would permit formal verification of the compliance at a later point of the MDSD process, e.g. at deployment- or runtime. Without the aspiration to be complete, we will then enumerate several other possible requirements. Finally we will, after these initial considerations, either propose a concrete modelling for the respective compliance concern or leave the subject open to further investigation. Although concrete proposals will be illustrated for some compliance concerns, it is important to point out that these models will serve as a starting point rather than be a final specification. Also advanced compliance concerns have to be covered in future.

4.2. Control flow

> *"The control flow compliance concern encompasses requirements concerning how things are done in business processes (i.e. what activities are carried out and in what order)."* [D2.1]

The control flow of business processes primarily consists of *activities*. These activities can be of basic nature; also structured activities such as loops or conditional activities are defined in common workflow languages such as BPEL. Activities are arranged within in a workflow; therefore they are given a certain order.

If we want to apply some kind of compliance on the control flow we should foremost concentrate on the *execution* of activities and the *order* of their execution.

Initial example and modelling approach:

A possible compliance requirement for a business process could be that a certain activity B must be executed after an activity A. This requirement could be addressed by a simple compliance model within the VbMF with a *MustExecuteAfter* class that contains two references to the *service* element from the VbMF *Core* model. While the first reference could specify the activity A, the second reference would define the activity B that must follow.

Some other possible requirements:

This kind of modelling, obviously, is rather limited if we consider other possible requirements towards compliance of the control flow such as:

- A certain process must execute an activity X before an activity Y and the execution of these activities does not have to be performed subsequently. Particularly there can be an activity Z in between the control flow of activity X and Z.
- A process must execute a certain activity at some point during its control flow.
- Two activities must not be executed by the same process.

Suggested approach:

We could easily consider these additional requirements and extend the conceptual model elements accordingly; however we believe that by using process fragments we can express the control flow compliance of a business process in a much more comprehensive way. We will thus show in [D4.3] how process fragments relate to COMPAS and the control flow compliance concern.

4.3. Locative

> *"The locative compliance concern addresses requirements concerning the location where business process activities are carried out."* [D2.1]

In contrast to the control flow compliance concern where we had to concentrate on the execution of activities and their order we now have to focus on the *location* of the execution. This section extends Section 3.3.3, by giving a more detailed introduction to locative compliance concerns.

We therefore have to elaborate how to identify the location of an execution. This usually is related to where a service is deployed at. A deployment model that e.g. specifies a host as a location for a service (execution) can also be used for expressing and validating the locative compliance concern. One possibility to determine the location would be to use the internet protocol (IP) address of a computer that executes an activity or business process as an identifier for the location. While an IP address would not directly reveal the location of a host, it could nonetheless be a unique identifier for a certain host. If we (just) want to bind the execution of e.g. a business process to a certain host, IP-based locative specification would be sufficient in order to specify a locative compliance concern. In contrast of using plain IP addresses for specifying hosts also hostnames of course can be used thanks to the worldwide applied domain name system (DNS). A DNS name already can encode some geographical information into its hostname as this is usually done with addresses of routers:

- at-vie15a-ra1-ae0-946.aorta.net (213.46.173.113)
- uk-lon01a-rd3-10ge-11-0.aorta.net (213.46.160.233)

More interestingly, hostnames can be associated with geographical location information using LOC records [RFC 1876]. Geo location information nonetheless is however only an example and not the only or primary locative compliance concern. We have seen that also host based locative information can help to specify the compliance of an execution effectively.

Initial example and modelling approach:

Because of valid contracts, some out-sourced activities of a business process have to be performed by a certain company. The company has registered a domain name company.test.com and all hosts of the company have hostnames within this domain.

We can model the requirement in VbMF as follows: An *ExecutionLocation* class with an attribute *domain* can be instantiated with the domain name of the company and referenced by an *ExecutionConcern* class, that references the process or process activity that must be executed on a host that domain name lies within *domain*.

Some other possible requirements:

- A certain business process consists of a computationally intensive activity that typically is performed by a grid provider. Because of legislative regulations, the transfer and execution of the according data e.g. may only be performed within a certain member state of the European Union.
- Business to business (B2B) Web service calls to certain countries may be forbidden because of existing international sanctions. While related but more appropriate to the focus on execution we could state that the execution of processes or process activities must not take place on or within a certain location.
- While we have until now focused on the execution of single processes or activities, another requirement could be that all execution has to take place within a certain country or company in order to eliminate risks concerning different legislations or unclear liabilities.

Suggested approach:

We suggest the following approach for modelling the locative compliance concern:

An *ExecutionCompliance* class consists of

- a reference, that specifies a *process* or *service* element of the VbMF *core* view.
- an instance of an *ExecutionLocation* that specifies one of the following possible information:
    - an IP address or IP address block
    - a host- or domain name
    - a country using an ISO 3166 alpha-2 code
    - geocoordinates as specified in Section 3 of [RFC 1876]
- a priority that may be instantiated with one of the following values
    - 1: must be executed on the location specified
    - 2, 3, 4,..: should be executed on the location specified with decreasing priority by increasing value.
    - -1: must not be executed on the location specified

- -2, -3, -4: should not be executed on the location specified with decreasing priority by increasing absolute value.

Notes:

As we in general have to deal with other parties, security becomes an important issue in regard to locative compliance concerns. How can we say that an IP address or hostname has not been faked or spoofed? As it is the case with the today's internet, we – like everyone – have to rely on underlying technologies like IPv4 or DNS. IPv6 and DNSSec could provide solutions in order to bring more security to the net, however: e.g. LOC records still would have to be verified. An external, trusted, device that would be connected to the host with the location in question may probably receive (signed) geo-coordinates via a GPS signal and could establish trust by cryptographic certificates but as not only these approaches would have to be investigated but also are out of scope of the COMPAS project we rather stop at this point, leave the related questions open for discussion and continue with the modelling of other compliance concerns.

4.4. Information

> *"The information compliance concern deals with the information used and produced in business processes as well as the syntax and semantics of this information."* [D2.1]

The topic of information is a broad subject and therefore we only illustrate some few possible compliance concerns in regard to information as used and produced in business processes. It also needs to be pointed out that this compliance concern may overlap with the security and privacy compliance concerns that have been categorized as advanced compliance concerns in [D2.1]. Therefore no security or privacy aspects in regard to information are covered here as the later compliance concerns particularly address questions in regard to information.

Initial example and modelling approach:

A business process defines the output of one of its process activities using XML schema definition (XSD) [XSD]. While it is possible to specify restrictions in XSD and therefore dictate the syntax of a valid XSD instance, the semantics of the information has to be checked independently. Does for example a passed identifier exist in a database?

Some other possible requirements:

- Compliance regulations may require information to be tagged e.g. by a serial.
- A domain specific language may be used as the in- or output of a process activity. The syntax and semantics of this information thus has to be checked.

Possible approaches:

For addressing the semantic concern:

Within the VbMF an *Information view model* covers the concern of data and business objects as used and produced by processes and process activities. We can thus use this view for e.g. applying compliance OCL [OCL] restrictions.

For addressing the syntax concern:

When dealing with XML data Document Schema Definition Languages (DSDL) like RELAX NG [vdV03] or Schematron [vdV07] would be interesting options for expressing restrictions in order to guarantee information compliance in regard to syntax as well. Another possibility how information compliance in regard to syntax could be achieved would be to use a plug-in

architecture: We could introduce a namespace attribute within a XSD complex type that is used for the corresponding data object. Such a namespace may indicate that encapsulated information is expressed in a certain language. A plug-in for the language that needs to be deployed within the compliance framework would then validate the syntax (and potentially the semantics) of the information.

4.5. Resource

> *"The resource compliance concern considers the question of which resources are used within business processes (e.g. employees and customers, and computerized systems)."* [D2.1]

While we already have covered an aspect of the resource compliance concern within the locative compliance concern, namely the execution of a business process or activity on a certain host, we particularly want to examine concerns that are more related to the field of quality of service within this section.

1. What stakeholders are involved or associated with a certain process or activity?
2. How much CPU cycles or memory does a (process) activity consume?

Regarding question one:

The BPEL4People extension for BPEL [B4P] permits a precisely defined mapping of people and processes and activities in the context of BPEL processes. This technology can thus be used to define and execute processes with human interaction. While [VieBOP] proposes a generic architecture for BPEL engines that could be used during runtime to execute BPEL4People processes, [HumanVbMF] presents conceptual models for human aspects of business processes together with a mapping to BPEL4People technology. We can reuse the models, namely the human view for also specifying the compliance of business processes. This seems to result in needless redundancy and indeed it introduces the question: are the respective models, once found in the modelling of the business process as well as the second one as recorded as a compliance concern consistent? The answer to this question is exactly the answer to the question if the modelled business process is compliant in regard to the modelled concern. Such a validation of models thus results in a direct test for compliance at design time where we can profit from already existing models that can be used for reference instances.

Regarding question two:

Various criteria for quantifying different computer resources exist. Amongst them are:

- Max memory
- Max CPU cycles
- Bandwidth Limits
- Download/Upload Volume

Initial example and modelling approach:

Within a certain business process an activity has to be performed manually by a qualified person that thus is authorized to perform this human task.

Within the human view of the VbMF we can annotate a certain process activity to be a human task. Also we can define the role of potential owners for this task and associate the respective person to this role. For expressing the compliance of a business process, that needs to

implement the process activity as a human task with the appropriate binding to human stakeholders, we would like to (re)use such an instance of a human view.

Suggested approach:

We already mentioned that we covered some of the compliance concerns that are related to resources within the locative compliance concerns. Regarding the human resources and stakeholders we propose the following new compliance class:

A *StakeholderCompliance* class consists of

- a reference to the VbMF *human* view, that specifies the human aspect compliance concerns of a business process.

Notes:

We have seen that we actually can reuse existing models for business processes for also expressing the compliance for a business process. When doing so the validation for compliance at design and deployment time becomes trivial. Usually however the compliance is determined at runtime using online or offline monitoring. In cases where it is possible and suffices to check for compliance at deployment time we could profit nonetheless from the following validation: ideally, only model instances need to be compared by identity: is the same model instance used for the process as it is required to implement according to the compliance? If not the same but an equal model instance would suffice, we would compare the compliance model instance with the process model instance by value. In such cases it might make sense not only to check for equality but also use comparisons that e.g. check for a subset, etc.

4.6. Temporal

> *"The temporal compliance concern takes requirements concerning when things are done/must not be done within a business process into account (e.g. in terms of relevant business events)."* [D2.1]

Initial example and modelling approach:

Within a business to business (B2B) scenario the computation of a certain third party Web service request could be more expensive during day than during night. For this reason not so urgent processes might be delayed during day and continued when the external Web service calls are scheduled for execution.

We could extend the above mentioned *ExecutionCompliance* class with an optional attribute *executionTime* that indicates the time for the execution of the process or process activity.

Another problem is mentioned in Section 2.1 of [D5.2]:

> *"Section 409 of SOX requires that a publicly traded company discloses information regarding material changes in the financial condition of the company in real-time. […] The requirement that relevant information be disclosed in "real-time" has so far commonly been interpreted as "within 2-4 business days"."*

There an execution of two up to four business days determines the compliance of an accordant process.

Some other possible requirements:

- The not so urgent execution of a process like backing up a system might take place during a dedicated interval. If the interval ended the process could be delayed and continued at a later time when e.g. the interval reoccurs again.
- A process or process activity must not be executed during a specific interval.

Suggested approach:

As the temporal compliance concern like the locative compliance concern deals with some kind of execution concern, we intend to extend the *ExecutionCompliance* class with a reference to a new *TemporalCompliance* class that consist of:

- An attribute that specifies the starting of an execution
- An attribute that specifies the end (or deadline) of an execution. It can be specified as:
  - the duration from the start of an execution (useful for the problem from Section 2.1 of [D5.2])
  - or can be a fixed date.
- A *reoccurrence* attribute may specify a period in terms of e.g. milliseconds
- *exceptions* may hold dates with exceptions to the specified reoccurrences.

4.7. Summary

We have examined different compliance concerns and proposed several modelling approaches for respective initial scenarios. We have enumerated some additional compliance requirements and covered these with some proposals. We would now like to summarize and bring together the different aspects of compliance concerns by proposing the following modelling approach for compliance concerns within the VbMF.

In compliance concerns we have seen a priority that specifies if e.g. an execution must, must not or should take place e.g. at a certain location. As this priority can be generalised we move this up to the level of a *ComplianceRequirement* that consists of various compliance concerns.

The *ComplianceRequirement* itself can be associated with a certain *ComplianceRule*[1] of a *ComplianceRegulation*[2] as well as a *ComplianceFramework*[3]. Additionally a *Risk* can be specified for the *ComplianceRequirement* that applies when such a compliance requirement is violated.

Various instances of a *ComplianceRequirement* can be logically combined by associating them within an instance of *ComplianceRequirements* (see Figure 12) that finally is associated with a process or process element by referencing an *Element* of the VbMF *core* model [HZD07]. Last but not least OCL constraints can be applied to *ComplianceRequirements* with its associated data in order to complete expressing of a particular compliance concern.

[1]Section 5 of [D2.1] covers Compliance Specification and introduces Compliance Rules or Policies that express Compliance Requirements.

[2] Examples for different Compliance Legislations or Regulations (Section 3.1 of [D2.1]) are: Basel II, Sarbanes-Oxley Act or Tabaksblat.

[3] Compliance Frameworks such as COSO or COBIT, have been introduced in Section 2 of [D2.1].

Figure 12 UML class for ComplianceRequirements

5. Conclusion

In this deliverable, we step from the high level perspective of the overall COMPAS architecture down into the details of the components that the architecture constitutes. We discuss these components' functionality, inter-relationships and give details of the technologies used in implementation. We also distinguish which components are to be newly developed, extended or reused. This gives an insight into the contributions that the COMPAS project provides technology-wise.

As the project intends to take advantage of some of the positive points of the MDSD paradigm, we also present in more detail the COMPAS model-driven integration component. This component consists of the extensible architecture. We clarify functionalities of prototypes, tools, and technologies used, extended or newly developed by COMPAS consortium, and elicit the relationships, the interactions between these components. Moreover, the document also presented the foundation of COMPAS model-driven integration architecture which comprises core modelling artefacts and supporting mechanisms provided in the extended view-based framework for modelling of process-driven systems and business compliance; part of the framework supports and , along with the concepts of domain specific languages to enable domain experts to represent those facilitated for effectively and productively eliciting compliance concerns as customised models. As such, the model-driven integration component serves as the basis for the interactions and integrations of prototypes, tools, and technologies from COMPAS WPs.

6. Reference documents

6.1. Internal documents

[DA.1] "Model-driven Integration Architecture for Compliance: Case Study and Technology Mappings, Evaluation Metrics", 2009-01-31

[DoW] "Description of Work" for COMPAS, final version of 2008-02-01.

[D2.1] "State-of-the-Art in the Field of Compliance Languages", 2008-07-31.

[D2.2] "Initial specification of compliance language constructs and operators", 2009-01-31.

[D3.1] "Specification of a behavioural model for services", 2009-01-31.

[D4.3] "Classification and specification of reusable process artefacts", 2010-07-31.

[D5.1] "State-of-the-Art in the Field of Adaptive Service Composition Monitoring and Management", 2008-07-30.

[D5.2] "Initial Goal-oriented data model", 2009-01-31.

6.2. External documents

[AAD+07a] A. Agrawal, M. Amend, M. Das, Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., and M. Zeller, Web Services Human Task (WS-HumanTask), Version 1.0; http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf.

[AAD+07b] Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A. and M. Zeller , "WS-BPEL Extension for People (BPEL4People)", Version 1.0; http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf.

[ADO00] W. van der Aalst, J. Desel and A. Oberweis, ed., *Business process management: Models, techniques, and empirical studies - Lecture Notes in Computer Science* (Vol. 1806). Springer-Verlag, 2000.

[AHK+03] W. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski and A. P. Barros, "Workflow patterns", *Distributed and Parallel Databases*, 14 (1), 2003, pp. 5-51.

[AKR+05] B. Axenath, E. Kindler and V. Rubin, "An open and formalism independent meta-model for business processes", *Proc. of the Workshop on Business Process Reference Models*, 2005, pp. 45–59.

[BASELII] Basel II Capital Accord, 2007; http://www.occ.gov/ftp/release/2007-123b.pdf

[B4P] http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/

[BPEL] IBM, BEA Systems, Microsoft, SAP AG, & Systems Siebel, "Business Process Execution Language for Web services", version 1.1; 2003, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

[BPM] Business Process Modelling Forum, "What is Business Process Modelling (BPM)?"; http://www.bpmodeling.com/faq/#faq_1

[BPMN] Business Process Modelling Notation (BPMN), 2006; http://www.bpmn.org

[EMF] Eclipse Modelling Framework; http://www.eclipse.org/emf/.

[EPC] E. Kindler, "On the semantics of EPCs: A framework for resolving the vicious circle", In Business Process Management, 2004, pp. 82–97.

[FBK+99] D. Ferraiolo, J. Barkley, D. R. Kuhn, "A role-based access control model and reference implementation within a corporate intranet". *ACM Trans. Information and System Security (TISSEC)*, 2(1), 1999, pp. 34-64.

[FRAG] The Frag Language, http://frag.sourceforge.net

[GJM91] C. Ghezzi and M. Jazayeri and D. Mandrioli, *Fundamentals of Software Engineering*, Prentice Hall, 1991.

[HTZ+08] T. Holmes, H. Tran, U. Zdun, and S. Dustdar, "Modeling Human Aspects of Business Processes - A View-Based, Model-Driven Approach", *4th European Conf. Model Driven Architecture Foundations and Applications (ECMDA'08)*, Springer LNCS, 2008, pp. 246-261.

[HZ06] C. Hentrich and U. Zdun, "Patterns for Process-Oriented Integration in Service-Oriented Architectures", *Proc. 11th European Conf. Pattern Languages of Programs*. (EuroPLoP 06), 2006, pp. 1-45.

[HZD07] H. Tran, U. Zdun and S. Dustdar, View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA. *In Int'l Conf. Business Process and Services Computing (BPSC'07)*, volume 116 LNI, 2007, pp. 105-124.

[HZD08a] H. Tran, U. Zdun, and S. Dustdar, View-based Integration of Process-driven SOA Models at Various Abstraction Levels. *Proc. 1st Int'l Workshop on Model-Based Software and Data Integration (MBSDI 2008)*, CCIS 8, Springer, 2008, pp. 55-66.

[HZD08b] H. Tran, U. Zdun and S. Dustdar, View-Based Reverse Engineering Approach for Enhancing Model Interoperability and Reusability in Process-Driven SOAs. *10th Int'l Conf. Software Reuse (ICSR'08)*, Springer LNCS, 2008, pp. 233-244.

[IEEE00] IEEE Recommended Practice for Architectural Description of Software Intensive Systems (Tech. Rep. No. IEEE-std-1471-2000), IEEE, 2000.

[MOF] OMG. Meta Object Facility (MOF™) 2.0, 2008; http://www.omg.org/spec/MOF/2.0/HTML

[MSDSL] Microsoft Domain-Specific Language (DSL) Tools, http://msdn.microsoft.com/en-us/library/bb126235(VS.80).aspx

[OAW] openArchitectureWare, 2002; http://www.openarchitectureware.org

[ODE] Apache (Orchestration Director Engine), 2008; http://ode.apache.org/.

[OMRDP] Open Distributed Processing Reference Model (IS 10746), 1998; http://isotc.iso.org/.

[OCL] http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL

[OSLO] MSDN Oslo Developer Center, http://msdn.microsoft.com/oslo

[RFC 1876] C. Davis, P. Vixie, T. Goodwin and I. Dickinson, *RFC 1876 - A Means for Expressing Location Information in the Domain Name System*, 1996

[SOX] Sarbanes-Oxley Act of 2002; http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=107_cong_public_laws&docid=f:publ204.107

[UML] Unified Modelling Language 2.0 (UML), 2004; http://www.uml.org

[vdV03] E. van der Vlist, *RELAX NG*, O'Reilly, 2003

[vdV07] E. van der Vlist, *Schematron*, O'Reilly, 2007

[VS06] M. Völter and T. Stahl, *Model-Driven Software Development: Technology, Engineering, Management,* Wiley, 2006.

[WSDL] Web Services Description Language 1.1; http://www.w3.org/TR/wsdl

[XML] XML Schema Part 1: Structures; http://www.w3.org/TR/xmlschema-1/ and Part 2: Datatypes http://www.w3.org/TR/xmlschema-2/, 2001

[XPDL] XML Process Definition Language (XPDL), 2005; http://www.wfmc.org/standards/XPDL.htm

[XSD] http://www.w3.org/XML/Schema