Using Model-Driven Views and Trace Links to Relate Requirements and Architecture: A Case Study

Huy Tran [1], Ta'id Holmes [1], Uwe Zdun [2], and Schahram Dustdar [1]

[1] Distributed Systems Group, Information Systems Institute, Vienna University of Technology, Austria, {htran, tholmes, dustdar}@infosys.tuwien.ac.at

[2] Software Architecture Group, Department of Distributed and Multimedia Systems, University of Vienna, Austria, uwe.zdun@univie.ac.at

Abstract Compliance in service-oriented architectures (SOA) means in general complying with laws and regulations applying to a distributed software system. Unfortunately, many laws and regulations are hard to formulate. As a result, several compliance concerns are realized on a per-case basis, leading to ad hoc, handcrafted solutions for each specific law, regulation, and standard that a system must comply with. This, in turn, leads in the long run to problems regarding complexity, understandability, and maintainability of compliance concerns in a SOA. In this book chapter, we present a case study in the field of compliance to regulatory provisions, in which we applied our view-based, model-driven approach for ensuring the compliance with ICT security issues in business processes of a large European company. The research question of this chapter is to investigate whether our model-driven, view-based approach is appropriate in the context of the case. This question is generally relevant, as the case is applicable to many other problem of requirements that are hard to specify formally (like the compliance requirements) in other business cases. To this end, we will present lessons learned as well as metrics for measuring the achieved degree of separation of concerns and reduced complexity.

1 Introduction

As the number of elements involved in an architecture grows, the complexity of design, development, and maintenance activities also extremely increases along with the number of the elements' relationships, interactions, and data exchanges – and becomes hardly manageable. We have studied this problem in the context of process-driven, service-oriented architectures (but observed similar problems in other kinds of architectures as well) (Tran et al., 2009a). Two important issues are (among other issues) reasons for this problem: First, the process descriptions comprise various tangled concerns, such as the control flow, data dependencies, service invocations, security, compliance, etc. This entanglement seriously reduces many aspects of software quality such as the *understandability*, *adaptability*, and *maintainability*. Second, the differences of language syntaxes and semantics, the difference of granularity at different abstraction levels, and the lack of explicit links between process design and implementation languages hinder the *reusability*, *understandability*, and *traceability* of software components or systems being built upon or relying on such languages.

In our previous work we introduced a novel approach for addressing the aforementioned challenges. Our approach exploits a combination of the concept of architectural views (IEEE, 2000) – a realization of the *separation of concerns* principle (Ghezzi et al., 2002) – and the model-driven development paradigm (MDD) (Stahl and Völter, 2006) – a realization of the *separation of abstraction levels*. This approach has been implemented in the View-based Modeling Framework – an extensible development framework for process-driven, service-oriented architectures (SOAs) (Tran et al., 2009a). In this chapter, we present a case study in the field of compliance to regulatory provisions in which we applied our approach for complying to ICT security issues in a business process of a large European banking company. In particular, the case study illustrates how our approach helps achieving the following major contributions: *first*, it captures different perspectives of a business process model in separated (semi-)formalized view models in order to adapt to various stakeholders' expertise; *second*, it links to the requirements of the system via a special requirements meta-data view formally modeling the parts of the requirements information needed in the model-driven architecture; *third*, it reduces the complexity of dependency management and enhances traceability in process development via explicit trace links between code, design, and requirements artifacts in the model-driven architecture. We also present lessons learned and preliminary quantitative evaluations on the case study to support the assessment of our approach regarding some aspects of software quality such as the understandability, adaptability, and maintainability.

The rest of the chapter is organized as follows. In Section 2 we introduce a working application scenario extracted from the business processes of an European banking company. Next, an overview of compliance in service-oriented architectures is provided in Section 3. Section 4 presents a qualitative analysis of our approach applied in the application scenario that illustrates how the aforementioned contributions can be achieved. The lessons learned and quantitative evaluations are provided in Section 5. We discuss the related work in Section 6 and conclude.

2 Case Study: A Loan Approval Process

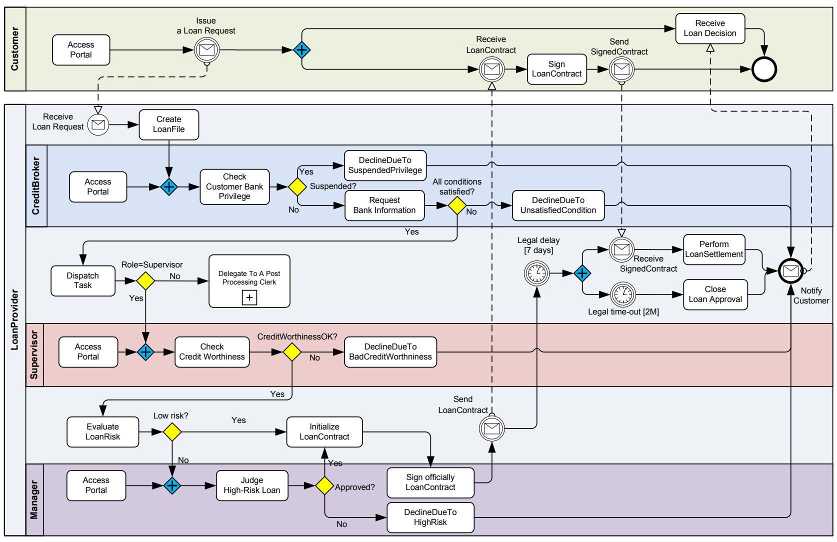


Fig. 11.1 Overview of the loan approval process

Throughout this study, we use a loan approval process of a large European banking company to illustrate the application of our approach in the domain of process-driven SOAs. The banking domain must enforce security and must be in conformity with the regulations in effect. Particular measures like separation of duties, secure logging of events, non-repudiable action, digital signature, etc., need to be considered and applied to fulfil the mandatory security requirements in order to comply with norms and standards of the banking domain as well as European laws and regulations. In particular, the company emphasizes the necessity of preventing the frauds, preserving the integrity of data, insuring a secure communication between the customers and the process, and protecting customer privacy. Figure 11.1 depicts the core functionality of the loan approval process by using BPMN [1] – a notational language widely used in industry for designing business processes.

At the beginning of the process, a credit broker is assigned to handle a new customer's loan request. He then performs preliminary inspections to ensure that the customer has provided valid credit information (e.g., saving or debit account). Due to the segregation of duties policy of the bank, the inspection carried out by the credit broker is not enough to provide the level of assurance required by the bank. If the loan enquired by the customer is less than one million euros, a post-processing clerk will take over the case. Otherwise, the case is escalated to a supervisor. In this stage, the customer's credit worthiness is estimated through a

[1] http://www.omg.org/spec/BPMN/1.1

larger set of data including sums of liabilities, sums of assets, third-party loans, etc. Finally, if no negative reports have been filed, the loan request is handed over to a manager who judges the loan risk and officially signs the loan contract. The customer shall receive either a loan declined notification or a successful loan approval.

3 Compliance in Process-Driven SOAs

Services are autonomous, platform-independent entities that can be described, published, discovered, and loosely coupled by using standard protocols (Papazoglou et al., 2008). Service-oriented architecture (SOA) is the main architectural style for service-oriented computing. In the scope of this chapter, we exemplify our approach for process-driven SOAs – a particular kind of SOAs utilizing processes to orchestrate services (Hentrich and Zdun, 2006) – because enterprizes increasingly use process-centric information systems to automate their business processes and services.

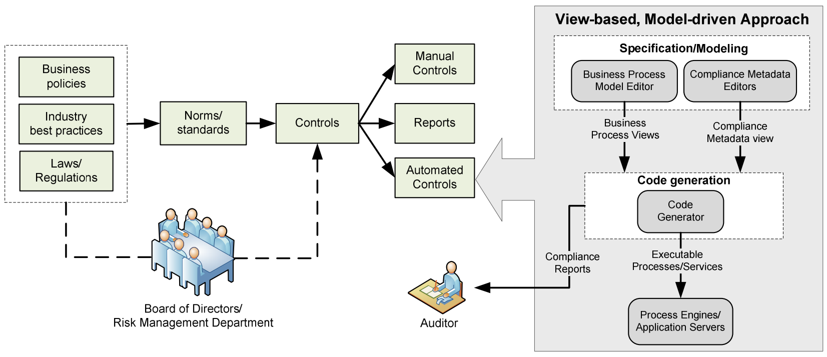


Fig. 11.2 Overview of the view-based, model-driven approach for supporting compliance in SOAs

Generally speaking, IT compliance means conforming to laws and regulations applying to an IT system such as the Basel II Accord [2], the Financial Security Law of France [3], the Markets in Financial Instruments Directive (MiFID) [4], and the Sarbanes-Oxley Act (SOX) [5]. These laws and regulations are designed to cover issues such as auditor independence, corporate governance, and enhanced financial disclosure. Nevertheless, laws and regulations are just one example of compliance

[2] http://www.bis.org/publ/bcbs107.htm
[3] http://www.senat.fr/leg/pjl02-166.html
[4] http://www.fsa.gov.uk/pages/About/What/International/mifid
[5] http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi
?dbname=107_cong_bills&docid=f:h3763enr.tst.pdf

concerns that might occur in process-driven SOAs. There are many other rules, policies, and constraints in a SOA that have similar characteristics. Some examples are service composition and deployment policies, service execution order constraints, information exchange policies, security policies, quality of service (QoS) constraints, and so on.

Compliance concerns stemming from regulations or other compliance sources can be realized using various *controls*. A control is any measure designed to assure a compliance requirement is met. For instance, an intrusion detection system or a business process implementing separation of duty requirements are all controls for ensuring systems security. As regulations are not very concrete on how to realize the controls, the regulations are usually mapped to established *norms* and *standards* describing more concretely how to realize the controls for a regulation. Controls can be realized in a number of different ways, including manual controls, reports, or automated controls (see Figure 11.2). Table 11.1 depicts some relevant compliance requirements that the company must implement in the loan approval process in order to comply with the applicable laws and regulations.

Compliance	Risks	Control
Order Approval	**R1**: Sales to fictitious customers are not prevented and detected	**C2**: Customer's identifications are verified with respect to identification types and information, customer's shipping and billing addresses are checked against some pre-defined constraints (countries, post code, phone number, etc).
Segregation of Duties (SoD)	**R2**: Duties are not properly segregated (SOX 404)	**C3**: The status of the account verification must be checked by a Financial Department staff. The customer's invoice must be checked and signed by a Sales Department staff.

Table 11.1 Compliance requirements for the loan approval process

4 View-Based, Model-Driven Approach: Overview

4.1 View-Based Modeling Framework

Our view-based, model-driven approach has been proposed for addressing the complexity and fostering the flexibility, extensibility, adaptability in process-driven SOA modeling development (Tran et al., 2009a). A typical business process in a SOA comprises various tangled concerns such as the control flow, data processing, service invocations, event handling, human interactions, transactions, to name but a few. The entanglement of those concerns increases the complexity of process-driven SOA development and maintenance as the number of involved services and processes grow. Our approach has exploited the notion of architectural views to describe the various SOA concerns. Each view model is a (semi)-formalized representation of a particular SOA or compliance concern. In other words, the view model specifies entities and their relationships that can appear in the corresponding view.

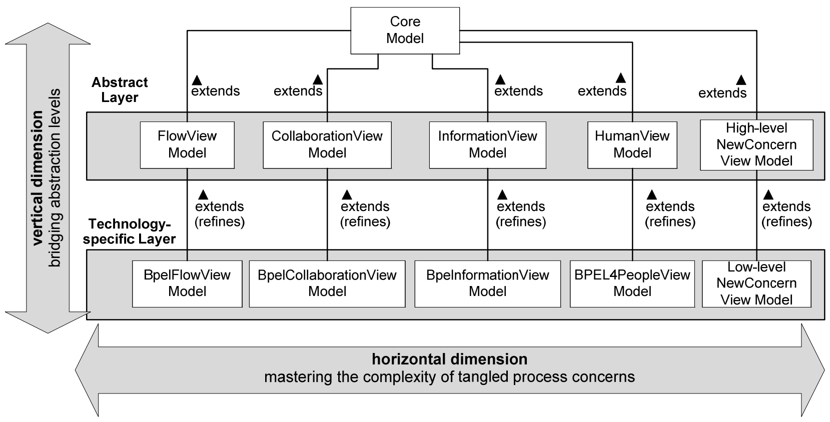

Core Model
Abstract Layer
extends
FlowView Model
CollaborationView Model
InformationView Model
HumanView Model
High-level NewConcern View Model
extends (refines)
Technology-specific Layer
BpelFlowView Model
BpelCollaborationView Model
BpelInformationView Model
BPEL4PeopleView Model
Low-level NewConcern View Model
vertical dimension
bridging abstraction levels
horizontal dimension
mastering the complexity of tangled process concerns

Fig. 11.3 Overview of the View-based Modeling Framework

Figure 11.3 depicts some process-driven SOA concerns formulated using VbMF view models. All VbMF view models are built upon the fundamental concepts of the Core model shown in Figure 11.4. Using the view extension mechanisms described in (Tran et al., 2009a), the developers can add a new concern by using a *New-Concern*-View model that extends the basic concepts of the Core model (see Figure 11.4) and defines additional concepts of that concern. The new requirements meta-data view, which is presented in Section 4.2, is derived using VbMF extension mechanisms for representing parts of the requirements information needed in process-driven SOAs and links them to the designs described using view models. As a result, the Core model plays an important role in our approach because it provides the basis for extending and integrating view models, and establishing and maintaining the dependencies between view models (Tran et al., 2009a, Tran et al., 2009b).

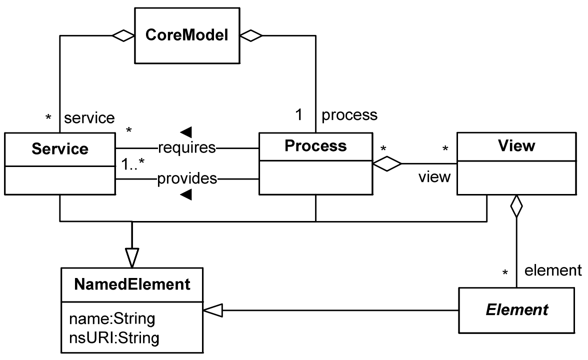


Fig. 11.4 Core model – the foundation for VbMF extension and integration

There are various stakeholders involved in process development at different levels of abstraction. For instance, business experts require high-level abstractions that offer domain or business concepts concerning their distinct knowledge and expertise while IT experts merely work with low-level, technology-specific descriptions. The MDD paradigm provides a potential solution to this problem by separating the platform-independent and platform-specific models (Stahl and Völter, 2006).

Leveraging this advantage of the MDD paradigm, VbMF has introduced a model-driven stack that has two basic layers: abstract and technology-specific. The abstract layer includes the views without the technical details such that the business experts can understand and manipulate them. Then, the IT experts can refine or map these abstract concepts into platform- and technology-specific views. For specific technologies, such as BPEL and WSDL, VbMF provides extension view models that enrich the abstract counterparts with the specifics of these technologies (Tran et al., 2009a). These extension views belong to the technology-specific layer shown in Figure 11.3.

Some activities during the course of process development may require information of multiple concerns, for instance, communications and collaborations between different experts, generation the whole process implementation, and so on. VbMF offered view integration mechanisms for combining separate views to provide a richer or a more thorough view of a certain process (Tran et al., 2009a). Finally, VbMF code generation can be used to produce executable process implementation and deployment configurations out of these views.

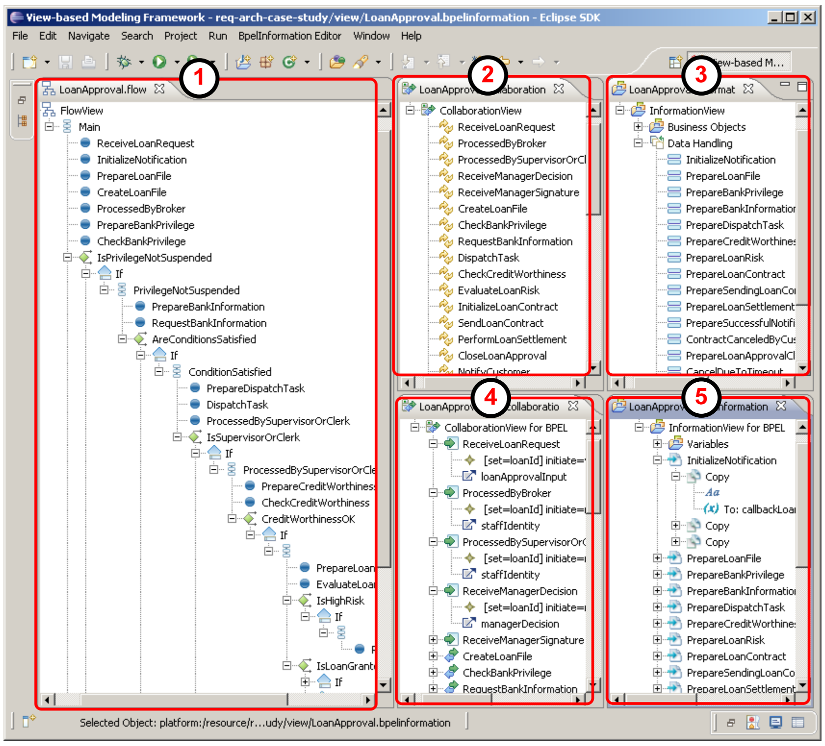
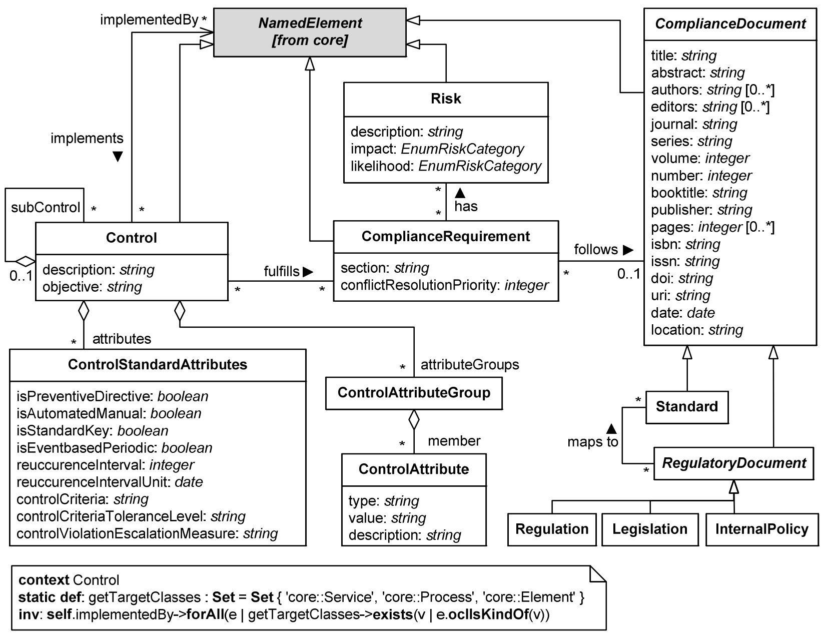


Fig. 11.4 The loan approval process development in VbMF: (1) The FlowView, (2-3) The high-level CollaborationView and InformationView, and (4-5) The low-level, technology-specific BpelCollaborationView and BpelInformationView

Figure 11.5 shows the loan approval process implemented using VbMF. These views are inter-related implicitly via the integration points from the Core model (Tran et al., 2009a). The detail of these views as well as their aforementioned relationships shall be clarified in Section 4.3 on the trace dependencies between VbMF views.

4.2 Linking to the Requirements: A Compliance Meta-data View

In this section, we present a Compliance Meta-data view for linking parts of the requirements and the design views of a SOA system. On the one hand, this view enables stakeholders such as business and compliance experts to represent compliance requirements originating from some compliance sources. On the other hand, it allows to annotate process-driven SOA elements described using VbMF (e.g., the ones shown in Figure 11.5) with the elicited compliance requirements. That is, we want to implement a compliance *control* for, e.g., a compliance regulation, standard, or norm, using a process or service.



```
context Control
static def: getTargetClasses : Set = Set { 'core::Service', 'core::Process', 'core::Element' }
inv: self.implementedBy->forAll(e | getTargetClasses->exists(v | e.oclIsKindOf(v))
```

Fig. 11.6 The Compliance Meta-data view model

The Compliance Meta-data view provides domain-specific architectural knowledge (AK) for the domain of a process-driven SOA for compliance: It describes which parts of the SOA, i.e., which services and processes, have which roles in the compliance architecture (i.e., *are they compliance controls?*) and to which compliance requirements they are linked. This knowledge describes important architectural decisions, e.g., why certain services and processes are assembled in a certain architectural configuration. In addition, the Compliance Meta-data view offers other useful aspects to the case study project: From it, we can automatically generate compliance documentation for off-line use (i.e., PDF documents) and for online use. Online compliance documentation is, for instance, used in monitoring applications that can explain the architectural configuration and rationale behind it, when a compliance violation occurs, making it easier for the operator to inspect and understand the violation.

A compliance requirement may directly relate to a process, a service, a business concern, or a business entity. Nonetheless compliance requirements not only introduce new but also depict orthogonal concerns to these: although usually related to process-driven SOA elements, they are often pervasive throughout the SOA and express independent concerns. In particular, compliance requirements can be formulated independently until applied to a SOA. As a consequence, compliance requirements can be *reused*, e.g., for different processes or process elements.

Figure 11.6 shows our proposed Compliance Meta-data view model. Annotation of specific SOA elements with compliance meta-data is done using compliance *Controls* that relate to concrete implementations such as a process or service (these are defined in other VbMF views). A *Control* often realizes a number of *ComplianceRequirements* that relate to *ComplianceDocuments* such as a *Regulation*, *Legislation*, or *InternalPolicy*. Such *RegulatoryDocuments* can be mapped to *Standards* that represent another types of *ComplianceDocument*. When a compliance requirement exists, it usually comes with *Risks* that arise from a violation of it. For documentation purposes, i.e., off-line uses, and for the implementation of compliance controls the *ControlStandardAttributes* help to specify general meta-data for compliance controls, e.g., if the control is automated or manual (*isAutomatedManual*). Besides these standard attributes, individual *ControlAttributes* can be defined for a compliance control within a certain *ControlAttributeGroup*.

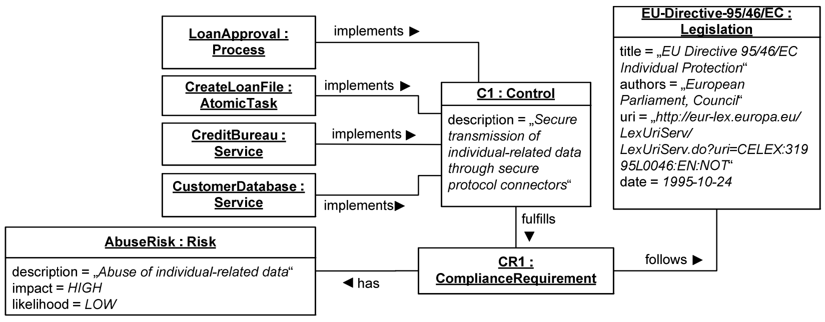

LoanApproval : Process
CreateLoanFile : AtomicTask
CreditBureau : Service
CustomerDatabase : Service
implements ▶
implements ▶
implements ▶
implements▶
C1 : Control
description = „Secure transmission of individual-related data through secure protocol connectors“
EU-Directive-95/46/EC : Legislation
title = „EU Directive 95/46/EC Individual Protection“
authors = „European Parliament, Council“
uri = „http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:NOT“
date = 1995-10-24
fulfills ▼
AbuseRisk : Risk
description = „Abuse of individual-related data“
impact = HIGH
likelihood = LOW
◀ has
CR1 : ComplianceRequirement
follows ▶

Fig. 11.7 Excerpt of the Compliance Meta-data view of the loan approval process

To provide for extensibility, we have realized a generic modeling solution: a *NamedElement* from the Core model can implement a *Control*. This way not only *Services* and *Processes* can realize a compliance control but as the View-based Modeling Framework is extended also other *NamedElements* can be specified to implement a *Control*. In order to restrict the arbitrary use, an OCL constraint is attached to the *Control* that can be adapted if necessary (i.e., the set of the getTargetClasses operation is extended with a new concept that can implement a *Control*).

Figure 11.7 shows an excerpt of the Compliance Meta-data view of the loan approval process that illustrates a directive from the European Union on the protection of individuals with regard to the processing of personal data. The compliance control *C1*, which fulfills the requirements *CR1*, is implemented by the elements of the loan approval process such as the process named *LoanApproval*, the task named *CreateLoanFile*, and the services named *CreditBureau* and *CustomerDatabase*. Those elements are modeled in VbMF as presented in Figures 1.5. The compliance requirement *CR1* follows the legislative document and is associated with an *AbuseRisk*.

In this way, the views in Figures 1.5 provide the architectural configuration of the processes and services whilst Figure 11.7 provides the compliance-related rationale for the design of this configuration. Using the Compliance Meta-data view, it is possible to specify compliance statements such as *CR1 is a compliance requirement that follows the EU Directive 95/46/EC on Individual Protection* [6] *and is implemented by the loan approval process* within VbMF.

The aforementioned information is useful for the project in terms of compliance documentation, and hence likely to be maintained and kept up-to-date by the developers and users of the system, because it can be used for generating the compliance documentation that is required for auditing purposes. If this documentation is the authoritative source for compliance stakeholders then it is also likely that they have an interest in keeping this information up to date. In doing so they may be further supported with, e.g., imports from other data sources. But in this model also important AK is maintained: In particular the requirements for the process and the services that implement the control are recorded. That is, this information can be used to explain the architectural configuration of the process and the services connected via a secure protocols connector. Hence, in this particular case this documented AK is likely to be kept consistent with implemented system and, at the same time, the rationale of the architectural decision to use secure protocol connectors does not get lost.

4.3 Model-Driven Traceability: Linking Architecture, Code, and Requirements

In the previous section we introduce the View-based Modeling Framework for modeling and developing processes using various perspectives that can be tailored for particular interests and expertise of the stakeholders at different levels of abstraction. We present in this section our view-based, model-driven traceability approach (VbTrace) realized as an additional dimension to the model-driven stack of VbMF (Tran et al., 2009b). VbTrace aims at supporting stakeholders in (semi-)automatically establishing and maintaining trace dependencies between the requirements, architecture, and implementations (i.e., process code artifacts) in VbMF (Tran et al., 2009b).

[6] http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:HTML

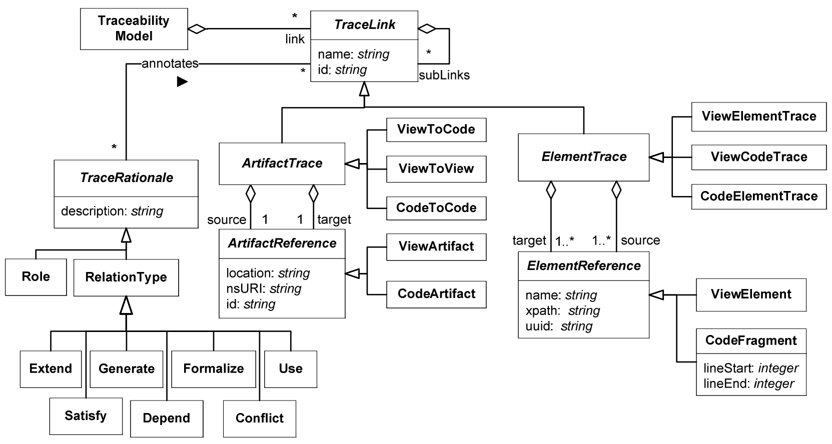


Fig. 11.8 The traceability view model

As we mentioned in Section 4.2, the relationships between the requirements and elements of a process represented in terms of VbMF views have been gradually established during the course of process development and stored in a Compliance Meta-data view. Now we elaborate how our traceability approach helps linking the various process views and code artifacts. The trace links between low-level, technology-specific views and code artifacts can be (semi-)automatically derived during the VbMF forward engineering process by using extended code generators or during the VbMF reverse engineering process by using extended view-based interpreters (Tran et al., 2009b). The relationships between a view and its elements are intrinsic while the relationships between different views are established and maintained according to the name-based matching mechanism for integrating and correlating views (cf. Tran et al., 2009a for more details).

Figure 11.8 presents the traceability view model – a (semi-)formalization of trace dependencies between development artifacts. The traceability view model is designed to be rich enough for representing trace relations from process design to implementation and be extensible for further customizations and specializations. There are two kinds of *TraceLinks* representing the dependencies at different levels of granularity: *ArtifactTraces* describing the relationships between artifacts such as view models, BPEL and WSDL files, and so on; *ElementTraces* describing the relationships between elements of the same or different artifacts such as view elements, BPEL elements, WSDL messages, XML Schema elements, and so forth. The source and target of an *ArtifactTrace* are *ArtifactReferences* that refers to the corresponding artifacts. *ElementTraces*, which are often sub-links of an *ArtifactTrace*, comprises several source and target *ElementReferences* pointing to the actual elements inside those artifacts. Each *TraceLink* might adhere to some *TraceRationales* that comprehend the existence, semantics, causal relations, or additional functionality of the link. The *TraceRationale* is open for extension and must be specialized later depending on specific usage purposes (Tran et al., 2009b).

In order to represent trace dependencies of the various view models at different levels of granularity, VbTrace has introduced three concrete types of *TraceLinks*: *ViewToViews* describe internal relationships of VbMF, i.e., relationships between view models and view elements, *ViewToCodes* elicit the traceability from VbMF to process implementations, and finally, *CodeToCodes* describe the relationships between the generated schematic code and the associated individual code. Along with these refined trace links between process development artifacts, we also extend the *ElementTrace* concept by fine-grained trace link types between elements such as *ViewElementTrace*, *ViewCodeTrace*, and *CodeElementTrace*. Last but not least, formal constraints in OCL have been defined in order to ensure the integrity and support the verification of the views instantiated from the traceability view model (Tran et al., 2009b). In the subsequent sections, we present a number of working scenarios to demonstrate how VbTrace can help establishing and maintaining trace dependencies.

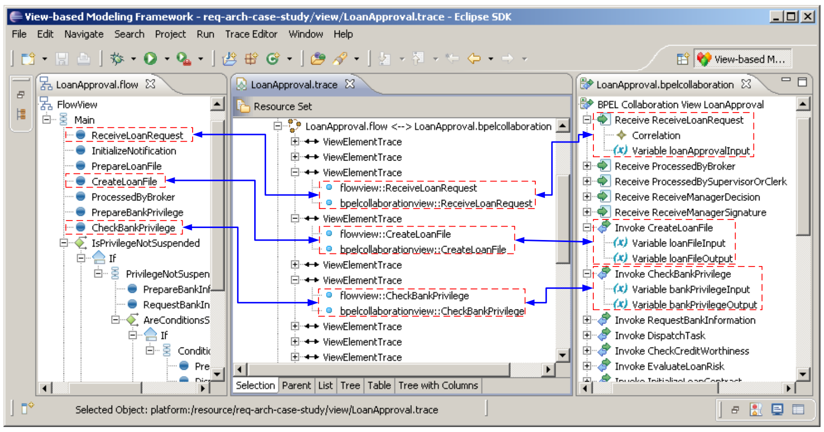

View-based Modeling Framework - req-arch-case-study/view/LoanApproval.trace - Eclipse SDK
File Edit Navigate Search Project Run Trace Editor Window Help
View-based M...
LoanApproval.flow
FlowView
Main
ReceiveLoanRequest
InitializeNotification
PrepareLoanFile
CreateLoanFile
ProcessedByBroker
PrepareBankPrivilege
CheckBankPrivilege
IsPrivilegeNotSuspended
If
PrivilegeNotSuspen
PrepareBankInf
RequestBankIn
AreConditionsS
If
Conditi
Pre
LoanApproval.trace
Resource Set
LoanApproval.flow <--> LoanApproval.bpelcollaboration
ViewElementTrace
ViewElementTrace
ViewElementTrace
flowview::ReceiveLoanRequest
bpelcollaborationview::ReceiveLoanRequest
ViewElementTrace
flowview::CreateLoanFile
bpelcollaborationview::CreateLoanFile
ViewElementTrace
ViewElementTrace
flowview::CheckBankPrivilege
bpelcollaborationview::CheckBankPrivilege
ViewElementTrace
ViewElementTrace
ViewElementTrace
Selection Parent List Tree Table Tree with Columns
LoanApproval.bpelcollaboration
BPEL Collaboration View LoanApproval
Receive ReceiveLoanRequest
Correlation
Variable loanApprovalInput
Receive ProcessedByBroker
Receive ProcessedBySupervisorOrClerk
Receive ReceiveManagerDecision
Receive ReceiveManagerSignature
Invoke CreateLoanFile
Variable loanFileInput
Variable loanFileOutput
Invoke CheckBankPrivilege
Variable bankPrivilegeInput
Variable bankPrivilegeOutput
Invoke RequestBankInformation
Invoke DispatchTask
Invoke CheckCreditWorthiness
Invoke EvaluateLoanRisk
Selected Object: platform:/resource/req-arch-case-study/view/LoanApproval.trace

Fig. 11.9 Illustration of trace links between the FlowView (left) and BpelCollaborationView (right) of the loan approval process

4.3.1 Traceability between VbMF views

As we mentioned in Section 4, the stakeholders might either formulate an individual view or communicate and collaborate with each other via combined views that provide richer or more thorough perspectives of processes (Tran et al., 2009a). For instance, a discussion between a business expert and an IT specialist might require the orchestration of the loan approval process activities along with the interactions between the process and other processes or services. The combination of the FlowView and either the CollaborationView or the BpelCollaborationView based on the name-based matching approach described in (Tran et al., 2007; Tran et al., 2009a) can offer such a perspective. Figure 11.9 illustrates the trace relationships of such combinations. The main purpose of view integration is to enhance the flexibility of VbMF in providing various tailored perspectives of the process representation. Because those perspectives might be used by the stakeholders for analyzing and manipulating the process model, we record the relationships raised from the above-mentioned combinations in the traceability according to specific stakeholders' actions and augment them with the *Dependency* type. For the sake of readability, we only present a number of selected trace dependencies and use the arrows to highlight the trace links stored in the traceability view.

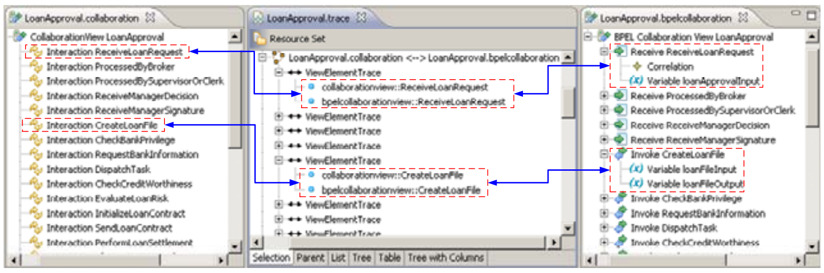
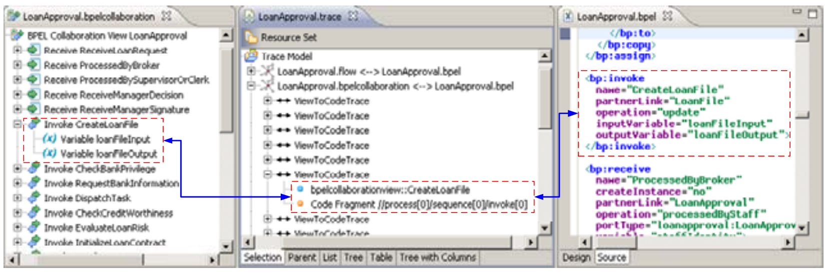

LoanApproval.collaboration
CollaborationView LoanApproval
Interaction ReceiveLoanRequest
Interaction ProcessedByBroker
Interaction ProcessedBySupervisorOrClerk
Interaction ReceiveManagerDecision
Interaction ReceiveManagerSignature
Interaction CreateLoanFile
Interaction CheckBankPrivilege
Interaction RequestBankInformation
Interaction DispatchTask
Interaction CheckCreditWorthiness
Interaction EvaluateLoanRisk
Interaction InitializeLoanContract
Interaction SendLoanContract
Interaction PerformLoanSettlement
LoanApproval.trace
Resource Set
LoanApproval.collaboration <--> LoanApproval.bpelcollaboration
ViewElementTrace
collaborationview::ReceiveLoanRequest
bpelcollaborationview::ReceiveLoanRequest
ViewElementTrace
collaborationview::CreateLoanFile
bpelcollaborationview::CreateLoanFile
Selection
Parent
List
Tree
Table
Tree with Columns
LoanApproval.bpelcollaboration
BPEL Collaboration View LoanApproval
Receive ReceiveLoanRequest
Correlation
Variable loanApprovalInput
Receive ProcessedByBroker
Receive ProcessedBySupervisorOrClerk
Receive ReceiveManagerDecision
Receive ReceiveManagerSignature
Invoke CreateLoanFile
Variable loanFileInput
Variable loanFileOutput
Invoke CheckBankPrivilege
Invoke RequestBankInformation
Invoke DispatchTask
Invoke CheckCreditWorthiness

Fig. 11.10 Illustration of trace links between the high- (left) and low-level CollaborationView (right) of the loan approval process

The refinements of high-level, abstract views to low-level, technology-specific ones are also recorded by using trace links of the type *ViewToView* to support the traceability between two view models as well as a number of *ViewElementTraces* each of which holds references to the corresponding view elements. Figure 11.10 shows an excerpt of the traceability view that consists of a number of trace links between the CollaborationView and BpelCollaborationView of the loan approval process.

4.3.2 Traceability between VbMF views and process implementations

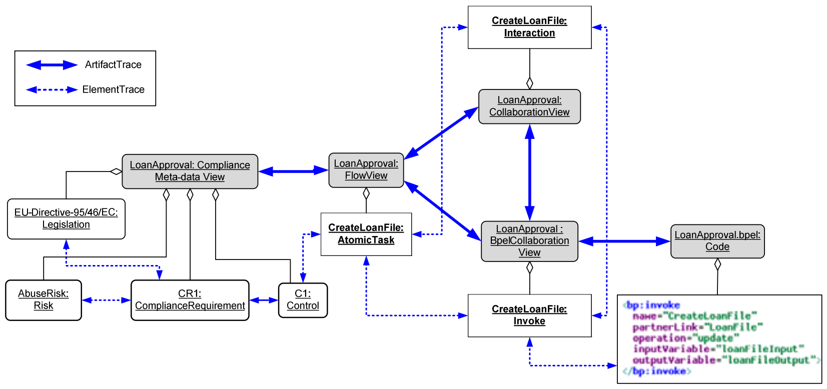

LoanApproval.bpelcollaboration
BPEL Collaboration View LoanApproval
Receive ReceiveLoanRequest
Receive ProcessedByBroker
Receive ProcessedBySupervisorOrClerk
Receive ReceiveManagerDecision
Receive ReceiveManagerSignature
Invoke CreateLoanFile
Variable loanFileInput
Variable loanFileOutput
Invoke CheckBankPrivilege
Invoke RequestBankInformation
Invoke DispatchTask
Invoke CheckCreditWorthiness
Invoke EvaluateLoanRisk
Invoke InitializeLoanContract
LoanApproval.trace
Resource Set
Trace Model
LoanApproval.flow <--> LoanApproval.bpel
LoanApproval.bpelcollaboration <--> LoanApproval.bpel
ViewToCodeTrace
bpelcollaborationview::CreateLoanFile
Code Fragment //process[0]/sequence[0]/invoke[0]
Selection
Parent
List
Tree
Table
Tree with Columns
LoanApproval.bpel
Design
Source

```
      </bp:to>
    </bp:copy>
  </bp:assign>

  <bp:invoke
    name="CreateLoanFile"
    partnerLink="LoanFile"
    operation="update"
    inputVariable="loanFileInput"
    outputVariable="loanFileOutput">
  </bp:invoke>

  <bp:receive
    name="ProcessedByBroker"
    createInstance="no"
    partnerLink="LoanApproval"
    operation="processedByStaff"
    portType="loanapproval:LoanApprov
```

Fig. 11.11 Illustration of trace links between the views (left) and generated BPEL code (right) of the loan approval process

The relationships between views and process implementation can be achieved in two different ways. On the one hand, process implementation are generated from the technology-specific views such as the BpelCollaborationView, BpelInformationView, etc., (Tran et al., 2009a). On the other hand, the view-based reverse engineering approach can also automatically extract process views from existing (legacy) implementations (Tran et al., 2008b). We recorded the trace links in both circumstances to maintain appropriate relationships between view models and process implementations to fully accomplish the traceability path from process designs to the implementation counterparts (see Figure 11.11).

4.3.3 An example of linking architectural views, code, and requirements


ArtifactTrace
ElementTrace
CreateLoanFile: Interaction
LoanApproval: CollaborationView
LoanApproval: Compliance Meta-data View
LoanApproval: FlowView
EU-Directive-95/46/EC: Legislation
CreateLoanFile: AtomicTask
LoanApproval : BpelCollaboration View
LoanApproval.bpel: Code
AbuseRisk: Risk
CR1: ComplianceRequirement
C1: Control
CreateLoanFile: Invoke

```
<bp:invoke
  name="CreateLoanFile"
  partnerLink="LoanFile"
  operation="update"
  inputVariable="loanFileInput"
  outputVariable="loanFileOutput">
</bp:invoke>
```

Fig. 11.12 Illustration of a traceability path from requirements through architectural views to code

We present a sample traceability path based on the traceability view established in the previous sections to illustrate how our traceability approach can support linking the requirements, architecture, and code (see Figure 11.12). The traceability path implies the trace links between the requirements and process elements – derived from the Compliance Meta-data view – followed by the relationships among VbMF views. The process implementation is explored at the end of the traceability path by using the trace dependencies between VbMF technology-specific views and process code.

Let us assume that there is a compliance requirement changed according to new regulations. Without our traceability approach, the stakeholders, such as business, domain, and IT experts, have to dig into the BPEL and WSDL code, identify the elements to be changed and manipulate them. This is time consuming and error-prone because there is no explicit links between the requirements to and process implementations. Moreover, the stakeholders have to go across numerous dependencies between various tangled concerns, some of which might be not relevant to the stakeholders expertise. Using our approach, the business and domain experts can better analyze and manipulate business processes by using the VbMF abstract views, such as the FlowView, CollaborationView, InformationView, Compliance Meta-data View, etc. The IT experts, who mostly work on either technology-specific views or process code, can better analyze and assess coarse-grained or fine-grained effects of these changes based on the traceability path.

5 Evaluation and Lessons Learned

So far we have presented a case study based on the development life cycle of an industrial business process that qualitatively illustrates the major contributions achieved by using our approach. To summarize, these are in particular: First, a business process model is (semi-)formally described from different perspectives that can be tailored and adapted to particular expertise and interests of the involving stakeholders. Second, parts of the requirements are explicitly linked to the system architecture and code by a special (semi-)formalized meta-data view. Third, our view-based traceability approach can help reducing the complexity of dependency management and improving traceability in process development. In addition, we also conducted a quantitative evaluation to support the assessment of our approach. The degree of separation of concerns and the complexity of business process models are measured because they are considered as the predictors of many important software quality attributes such as the understandability, adaptability, maintainability, and reusability (Fenton and Pfleeger, 1997). This evaluation focuses on the view-based approach as the foundation of our approach and provides evidence supporting our claims regarding the above-mentioned software quality attributes.

5.1 Evaluation

5.1.1 Complexity

In practice, there are several efforts aiming at quantifying the complexity of software such as Line of Code (Fenton and Pfleeger, 1997), McCabe complexity metrics (McCabe, 1976), Chidamber-Kemerer metrics (Chidamber and Kemerer, 1994), etc. However, Lange, 2006 suggested that these metrics are not suitable for measuring the complexity of MDD artifacts. Lange, 2006 proposed an approach for measuring model size based on the four dimensions of (Fenton and Pfleeger, 1997). Lange's metric is of cognitive complexity that rather reflects the perception and understanding of a certain model from a modeler's point of view (Fenton and Pfleeger, 1997). That is, the higher the size complexity, the harder it is to analyze and understand the system (Fenton and Pfleeger, 1997). The complexity used in our study is a variant of Lange's model size metrics (Lange, 2006), which is extended to support specific concepts of process-driven SOAs and the MDD paradigm. It measures the complexity based on *the number of the model's elements and the relationships between them*.

Process	**VbMF(Hi)**			**VbMF(Lo)**		**IntegrationPoint**		**Process impl.**
	FV	CV	IV	BCV	BIV	IP_{high}	IP_{low}	BPEL/WSDL
Travel Agency (TAP)	33	33	43	56	261	17	40	355
Order Handling (OHP)	29	36	44	65	285	17	46	383
Billing Renewal (BRP)	81	63	85	132	492	48	177	700
CRM Fulfilment (CFP)	49	74	78	131	535	31	88	730
Loan Approval (LAP)	68	44	48	104	651	34	85	871

Table 11.2 The complexity of process descriptions and VbMF views

In addition to the loan approval process (LAP) presented in Section 2, we perform the evaluation of complexity on four other use cases extracted from industrial process including a travel agency process (TAP) from the domain of tourism, an order handling process (OHP) from the domain of online retailing, a billing renewal process (BRP) and a CRM fulfillment process (CFP) from the domain of Internet service provisioning. We apply the above-mentioned model-based size metric for each main VbMF view such as the FlowView (FV), high-level and low-level CollaborationViews (CV/BCV), and high-level and low-level InformationViews (IV/BIV). Even though the correlation of views are implicit performed via the name-based matching mechanism (Tran et al., 2009a), the name-based integration points between high-level views (IP_{high}) and low-level views (IP_{low}) are calculated because these indicates the cost of separation of concerns principle realized in VbMF. Table 11.2 shows the comparison of these metrics of VbMF views to those of process implementation in BPEL technology, which is widely used in practice for describing business processes. Note that the concerns of process implementation are not naturally separated but rather intrinsically scatted and tangled. We apply the same method to calculate the size metric of the process implementation based on its elements and relationships with respect to the corresponding concepts of VbMF views.

The results show that the complexity of each of VbMF views is lower than that of the process implementation. Those results prove that our approach has reduced the complexity of business process model by the notion of (semi-)formalized views. We also measure a high-level representation of process by using an integration of VbMF abstract views and a low-level representation of process by using an integration of VbMF technology-specific views. The numbers say that the complexity of the high-level (low-level) representation is much less than (comparable to) that of the process implementation. The overhead of integration points occurs in both aforementioned integrated representations.

5.1.2 Separation of Concerns

To assess the separation of concerns, we use the Process-driven Concern Diffusion metric (PCD), which is derived from the metrics for assessing the separation of concerns in aspect-oriented software development proposed in (Sant'Anna et al., 2003). The PCD of a process concern is a metric that counts the number of elements of other concerns which are either tangled in that concern or directly referenced by elements of that concern. The higher the PCD metric of a concern, the more difficult it is for the stakeholders to understand and manipulate the concern. The measurement of PCD metric in all processes mentioned in Section 5.1.1 are presented in Table 11.3.

Process (%)	Flow			Collaboration			Information Flow		
	Without VbMF	With VbMF	Reduced (%)	Without VbMF	With VbMF	Reduced (%)	Without VbMF	With VbMF	Reduced (%)
TAP	175	17	90.3	186	40	78.5	85	23	72.9
OHP	212	17	92.0	221	46	79.2	93	29	68.8
BRP	411	48	88.3	409	117	71.4	195	69	64.6
CFP	398	31	92.2	407	88	78.4	176	57	67.6
LAP	425	34	92.0	431	68	84.2	188	69	63.3

Table 11.3 Measures of Process-driven Concern Diffusion

A process description specified using BPEL technology often embodies several tangled process concerns. VbMF, by contrast, enables the stakeholders to formulate the process through separate view models. For instance, a process control-flow is described by a BPEL description that often includes many other concerns such as service interactions, data processing, transactions, and so on. As a result, the diffusion of the control-flow concern of the process description is higher than that of the VbMF FlowView. The results show that the separation of concerns principle exploited in our view-based, model-driven approach has significantly reduced the scatter and tanglement of process concerns. We have achieved a significant decrement of the diffusion of the control-flow approximately of 90%, which denotes a better understandability and maintainability of the core functionality of processes. For other concerns, our approach is also shown to notably reduce concern diffusions by roughly 80% for the collaboration concern and about 60% for the information concern, and therefore, improve the understandability, reusability, and maintainability of business process models.

5.2 Lessons Learned

The quickly increasing of the complexity of design, development, and maintenance activities and maintenance due to the thriving of the number of elements involved in an architecture is very challenging in the context of process-driven, service-oriented architectures. We also observed similar problems in other kinds of architectures that expose the following common issues. First, a system description, e.g., an architectural specification, a model, etc., embodies various tangled concerns. As a consequence, the entanglement seriously reduces many aspects of software quality such as the *understandability*, *adaptability*, and *maintainability*. Second, the differences of language syntaxes and semantics, the difference of granularity at different abstraction levels, and the lack of explicit links between these languages hinder the *understandability* and *traceability* of software components or systems being built upon or relying on such languages. Last but not least, parts of the system's requirements are hard to specify formally, for instance, the compliance requirements. These intrinsic issues (among other issues) are ones of reasons which impede the correlating of requirements and the underlying architectures.

Our study showed that it is feasible to facilitate a view-based, model-driven approach to overcome the aforementioned challenges. Our approach enables flexible, extensible (semi-)formalized methods to represent the software system using separate architectural views. The flexibility and extensibility of our approach have been confirmed including the devising and using an additional model-driven requirement view for adding AK meta-data with reasonable effort and a traceability view for supporting establishing and maintaining dependency relationships between the architecture and the corresponding implementations. In particular, this study also provided evidences to confirm that it is possible in the context of a project to record specific AK that is domain-specifically relevant for a project using such a view.

Moreover, the model-driven approach complemented by the traceability view model can help to keep the data in the AK view up-to-date and *consistent* with the project. As a result, the integrity and consistency of the links from requirements to architecture and code can be maintained. To this end, it is reasonable to connect the data recorded in the AK view with other meta-data that needs to be recorded in the project anyway. This would be an additional incentive for developers to document the AK. In our study, compliance in service-oriented systems is illustrated as an area where this is feasible because a lacking or missing compliance documentation can lead to severe legal consequences. Nonetheless, our general approach can also be applied for custom AK without such additional incentives.

There is a limitation in our approach that only specific AK – linked to a domain specific area like compliance – is recorded and other AK might get lost. It is the responsibility of a project to make sure that all relevant AK for understanding an architecture gets recorded. In addition, our view-based, model-driven exploits the notion of architectural views – a realization of the separation of concern principle – and the MDD paradigm – a realization of the separation of levels of abstraction. In particular, the system description is managed and formulated through separate view models that are integrated and correlated via the name-based matching mechanism (Tran et al., 2009a).

On the one hand, this supposes additional investments and training are required at the beginning for instantiating view models, traceability models, and model transformation rules. In case of legacy process-driven SOAs, the view-based reverse engineering might partially help stakeholders quickly coming up with view models extracted from the existing business process descriptions. However, manual interventions of stakeholders are still needed to analyze and improved the extracted views, and sometimes, the corresponding the traceability models. On the other hand, this also implies that, comparing to a non-view-based or non-model-driven approach, additional efforts and tool supports are necessitated for managing the consistency of views and traceability models as those can be manipulated by different stakeholders as well as enabling change propagation among them. Nonetheless, the maintenance of trace dependencies between views can be enhanced by using hierarchical or ontology-based matching and advanced trace link recovery techniques (Antoniol et al., 2002).

However, the project can benefit in the long term regarding the reducing maintenance cost due to the enhancement of understandability, adaptability, and traceability as well as the preserved consistent AK. Nonetheless, it is possible to introduce our approach into a non-model-driven project (e.g., as a first step into model-driven development). For doing this, at least a way to identify the existing architectural elements, such as components and connectors, must be found. But this would be considerably more work than adding the view to an existing model-driven project.

6 Related Work

In our study, we applied our prior works that is the view-based, model-driven approach for process-driven SOAs (Tran et al., 2007; Tran et al., 2008a; Tran et al., 2008b; Tran et al., 2009a; Tran et al., 2009b) in the field of compliance to regulatory provisions. Therefore, more in-depth comparisons and discussions on the related work of the view-based, model-driven approach can be found in (Tran et al., 2007; Tran et al., 2008a; Tran et al., 2008b; Tran et al., 2009a) those of the name-based view integration mechanism can be found in (Tran et al., 2010), and those of the view-based traceability approach can be found in (Tran et al., 2009b). To this end, we merely discuss the major related works in the area of bridging requirements, architecture, and code.

A number of efforts provide modeling-level viewpoint models for software architecture (Rozanski and Woods, 2005), 4+1 view model by Kruchten, 1995 and the IEEE 1471 standard (IEEE, 2000) concentrating on various kinds of viewpoints. While some viewpoints in these works and VbMF overlap, a general difference is, that VbMF operates at a more detailed abstraction level – from which source code can be generated via MDD. Previous works on better support for codifying the AK have been done in the area of architectural decision modeling. Jansen et al., 2007 see software architecture as being composed of a set of design decisions. They introduce a generic meta-model to capture decisions, including elements such as problems, solutions, and attributes of the AK. Another generic meta-model that is more detailed has been proposed by Zimmermann et al, 2007. Tyree and Ackerman, 2005 proposed a highly detailed, generic template for architectural decision capturing.

De Boer et al, 2007 propose a core model for software architectural knowledge (AK). This core model is a high-level model of the elements and actions of AK and their relationships. In contrast to that, our models operate at a lower-level – from which code can be generated (the core model in VbMF is mainly defining integration points for MDD). Of course, the core model by de Boer et al.\ and VbMF could be integrated by providing the model by de Boer et al.\ as a special AK view in VbMF that is linked to the lower-level VbMF models via the matching mechanisms and trace links discussed in this paper.

Question, Options, and Criteria diagrams raise a design question, which points to the available solution options, and decision criteria are associated with the options (MacLean et al., 1991). This way decisions can be modeled as such. Kruchten et al., 2006 extend this research by defining an ontology that describes the information needed for a decision, the types of decisions to be made, how decisions are being made, and their dependencies. Falessi et al., 2006 present the Decision, Goal, and Alternatives framework to capture design decisions. Recently, Kruchten et al., 2009 extended these ideas with the notion of an explicit decision view – akin to the basic view-based concepts in our approach.

Wile, 2001 introduced a runtime approach that focuses on monitoring running systems and validating their compliance with the requirements. Grünbacher et al., 2003 proposed an approach that facilitates a set of architectural concepts to reconcile the mismatches between the concepts of requirements and those of the corresponding architectures. Hall et al., 2002 proposed an extension to the problem-frames approach to support the iteration between problem and solution structures in which architectural concepts can be considered as parts of the problem domain rather than the solution domain. Heckel and Engels, 2002 proposed an approach to relate functional requirements and software architecture in order to arrive at a consistent overall model in which a meta model is facilitated to provide separate packages for the functional requirements and the architectural view and a third package representing the relation between these views.

In contrast to our work, most of the related work on architectural decision modeling focus on *generic* knowledge capturing. Our approach proposes to capture AK in a *domain-specific* fashion as needed by a project. Hence in our work some AK is not as explicit as in the other approaches. For example, the collaborations of components are shown in the CollaborationView whereas the other approaches rather use a typical component and connector view. The decision drivers and consequences of the decisions are reported in the compliance sources and as risks. That means, our domain-specific AK view adopts the terminology from the compliance field, and it must be mapped to the AK terminology in order to understand the overlaps. None of the related works provide detailed guidelines how to support the AK models or views through MDD. On the contrary, this is a focus of our work. Additionally, using the model-driven development paradigm in our approach gains a twofold advantage. On the one hand, stakeholders working at different abstraction levels are offered tailored perspectives according to their expertise and interests. On the other hand, data in the AK view are preserved and keeping up-to-date and *consistent* with other parts of the project.

7 Conclusion

In this book chapter we presented an approach for relating requirements and architecture using model-driven views and automatically generated trace links. We demonstrated the applicability of this approach in the context of a case study in the field of ICT security compliance. The results suggest that using our approach it is possible to describe a business process in (semi-)formal way from different perspectives that can be tailored and adapted to particular expertise and interests of the involved stakeholders. Our quantitative evaluation gives evidence that this approach also has benefits in terms of reduced complexity and concern diffusion. Using a special (semi-)formalized meta-data view, we were able to link parts of the requirements to the system architecture described by these views and the code generated from them. In this context, our view-based traceability approach supports the automated dependency management and hence improves the traceability in process development. Our ongoing work is to complement this framework with an integrated development environment that facilitates collaborative model-driven design with different stakeholders as well as a runtime governance infrastructure that enacts the detection of compliance violations and compliance enforcement according to the monitoring directives generated from compliance DSLs and the Compliance Meta-data view model.

Acknowledgments The authors would like to thank the anonymous reviewers for providing constructive, insightful comments that greatly help to improve this chapter. This work was supported by the European Union FP7 project COMPAS, grant no. 215175.

References

Antoniol, Giuliano, Canfora, Gerardo, Casazza, Gerardo, Lucia, Andrea De, and Merlo, Ettore (2002). Recovering traceability links between code and documentation. IEEE Trans. Softw. Eng., 28(10):970–983.

Boer, Remco C. De, Farenhorst, Rik, Lago, Patricia, van Vliet, Hans, Clerc, Viktor, and Jansen, Anton (2007). Architectural knowledge: getting to the core. In Quality of software architectures (QoSA), pages 197–214.

Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Trans. Softw. Eng., 20(6):476–493.

Falessi, D., Becker, M., and Cantone, G. (2006). Design decision rationale: Experiences and steps towards a more systematic approach. SIG-SOFT Software Eng. Notes 31 – Workshop on Sharing and Reusing Architectural Knowledge, 31(5).

Fenton, Norman and Pfleeger, Shari L. (1997). Software metrics (2nd ed.): a rigorous and practical approach. PWS Publishing Co., Boston, MA, USA.

Ghezzi, Carlo, Jazayeri, Mehdi, and Mandrioli, Dino (2002). Fundamentals of Software Engineering, 2nd Edition. Prentice Hall.

Grünbacher, Paul, Egyed, Alexander, and Medvidovic, Nenad (2003). Reconciling software requirements and architectures with intermediate models. J. Softw. & Syst. Model., 3(3):235–253.

Hall, J. G., Jackson, M., Laney, R. C., Nuseibeh, B., and Rapanotti, L. (2002). Relating software requirements and architectures using problem frames. In IEEE Int'l Conf. Requirements Engineering, pages 137–144. IEEE Comp. Soc.

Heckel, R. and Engels, G. (2002). Relating functional requirements and software architecture: separation and consistency of concerns. J. Software Maintenance and Evolution: Research and Practice, 14(5):371–388.

Hentrich, Carsten and Zdun, Uwe (2006). Patterns for Process-Oriented Integration in Service-Oriented Architectures. In 11th European Conf. Pattern Languages of Programs (EuroPLoP 2006), pages 1–45, Irsee, Germany.

IEEE (2000). Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE-Std-1471-2000.

Jansen, A. G. J., van der Ven, J., Avgeriou, P., and Hammer, D. K. (2007). Tool support for architectural decisions. In 6th IEEE/IFIP Working Conf. Software Architecture (WICSA), Mumbai, India.

Kruchten, P., Lago, P., and Vliet, H. van (2006). Building up and reasoning about architectural knowledge. In QoSA 2006 (Vol. LNCS 4214), pages 43–58.

Kruchten, Philippe (1995). The 4+1 view model of architecture. IEEE Softw., 12(6):42–50.

Kruchten, Philippe, Capilla, Rafael, and Duenas, Juan Carlos (2009). The decision view's role in software architecture practice. IEEE Software, 26:36–42.

Lange, Christian F. J. (2006). Model size matters. In Models in Software Engineering, Workshops and Symposia at MoDELS 2006, LNCS, pages 211–216. Springer-Verlag.

MacLean, A., Young, R. M., Bellotti, V., and Moran, T. (1991). Questions, options, and criteria: Elements of design space analysis. Human-Computer Interaction, 6(3–4):201–250.

McCabe, Thomas J. (1976). A complexity measure. IEEE Trans. Softw. Eng., 2(4):308–320.

Papazoglou, Mike P., Traverso, Paolo, Dustdar, Schahram, and Leymann, Frank (2008). Service-oriented computing: a research roadmap. Int'l J. Cooperative Inf. Syst., 17(2):223–255.

Rozanski, Nick and Woods, Eoin (2005). Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley.

Sant'Anna, Cláudio, Garcia, Alessandro, Chavez, Christina, Lucena, Carlos, and v. von Staa, Arndt (2003). On the reuse and maintenance of aspect-oriented software: An assessment framework. In XVII Brazilian Symposium on Softw. Eng.

Stahl, Thomas and Völter, Markus (2006). Model-Driven Software Developoment. John Wiley & Sons.

Tran, Huy, Holmes, Ta'id, Zdun, Uwe, and Dustdar, Schahram (2009a). Modeling Process-Driven SOAs – a View-Based Approach, chapter 2. IGI Global, Handbook of Research on Business Process Modeling edition.

Tran, Huy, Zdun, Uwe, and Dustdar, Schahram (2007). View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA. In Int'l Conf. Business Process and ServicesComputing (BPSC), volume 116 of LNI, pages 105–124. GI.

Tran, Huy, Zdun, Uwe, and Dustdar, Schahram (2008a). View-based Integration of Process-driven SOA Models At Various Abstraction Levels. In First Int'l Workshop on Model-Based Software and Data Integration MBSDI 2008, pages 55–66. Springer.

Tran, Huy, Zdun, Uwe, and Dustdar, Schahram (2008b). View-Based Reverse Engineering Approach for Enhancing Model Interoperability and Reusability in Process-Driven SOAs. In 10th Int'l Conf.Software Reuse, ICSR, pages 233–244. Springer.

Tran, Huy, Zdun, Uwe, and Dustdar, Schahram (2009b). VbTrace: Using View-based and Model-driven Development to Support Traceability in Process-driven SOAs. J. Soft. & Syst. Model. doi:10.1007/s10270-009-0137-0.

Tran, Huy, Zdun, Uwe, and Dustdar, Schahram (2010). Name-based view integration for enhancing the reusability in process-driven SOAs. In 1st Int'l Workshop on Reuse in Business Process Management (rBPM) at BPM 2010, pages 1–12. Springer-Verlag.

Tyree, J. and Ackerman, A. (2005). Architecture decisions: Demystifying architecture. IEEE Software, 22(19–27).

Wile, D. S. (2001). Residual requirements and architectural residues. In Fifth IEEE Int'l Symposium on Requirements Engineering, pages 194–201. IEEE Comput. Soc.

Zimmermann, O., Gschwind, T., Kuester, J., Leymann, F., and Schuster, N. (2007). Reusable architectural decision models for enterprise application development. In Quality of Software Architecture (QoSA) 2007.

Biographies

Huy Tran is a post-doctoral researcher working at the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Austria. He received his bachelor degree in Computer Science and Engineering from Ho Chi Minh City University of Technology, Vietnam in *2002* and his doctoral degree in Computer Science specialized in Software Engineering at Vienna University of Technology, Austria in December *2009*. His current research interests include: software architecture, model-driven engineering, domain-specific modeling, business processes, compliance engineering and service-oriented computing.

Ta'id Holmes studied Computer Science at the Vienna University of Technology and Organic Chemistry at the Vienna University of Technology, CPE Lyon and Université Claude Bernard Lyon 1. He received a Dipl.-Ing. from the Vienna University of Technology in Software Engineering/Internet Computing and a DEA (Diplôme d'Études Approfondies) in Chimie Organique Fine from the Université Claude Bernard Lyon 1. Since March 2007 he is a guest lecturer at the Vienna University of Economics and Business Administration. In May 2007 he joined the Distributed Systems Group of the Institute of Information Systems at the Vienna University of Technology as a research assistant.

Uwe Zdun is a full professor for software architecture at the Faculty of Computer Science, University of Vienna. He received his doctoral degree from the University of Essen in 2002. His research focuses on architectural decision, software patterns, modelling of complex software systems, service-oriented systems, domain-specific languages, and model-driven development. Uwe has published more than 100 articles in journals, workshops, and conferences, and is co-author of the books "Remoting Patterns – Foundations of Enterprise, Internet, and Real-time Distributed Object Middleware" (J. Wiley & Sons) and "Software-Architektur: Grundlagen, Konzepte, Praxis" (Elsevier/Spektrum). He is European Editor of the journal Transactions on Pattern Languages of Programming (TPLoP) published by Springer, and Associate Editor-in-Chief for design and architecture for the IEEE Software magazine.

Schahram Dustdar is a Full Professor for Internet Technologies at the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology (TU Wien) where he is director of the Vita Lab and Honorary Professor of Information Systems at the Department of Computing Science at the University of Groningen (RuG), The Netherlands. He received his M.Sc. (1990) and PhD. degrees (1992) in Business Informatics (Wirtschaftsinformatik) from the University of Linz, Austria. In April 2003 he received his habilitation degree (venia docendi).