Model-driven and Domain-specific Architectural Knowledge View for Compliance Meta-data in Process-driven SOAs

Ta'id Holmes, Huy Tran, Uwe Zdun, and Schahram Dustdar
Distributed Systems Group
Institute of Information Systems
Vienna University of Technology
Vienna, Austria
{tholmes, htran, zdun, dustdar}
@infosys.tuwien.ac.at

ABSTRACT

Architectural knowledge tends to get lost as the architecture evolves. In many cases, the main reason is that there are no incentives for stakeholders to invest enough time into recording the architectural knowledge. This is in part due to the generic nature of architectural knowledge recording and sharing means, such as architectural decision templates and meta-models. In this paper, we investigate on the feasibility of a domain-specific architectural knowledge view in the context of a model-driven project. The domain-specific approach helps us to make architectural knowledge (AK) recording more useful for a project apart from the goal of AK sharing and reuse. Model-driven development helps us to ensure the consistency of the architectural knowledge as it is part of the generation process. Finally, depicting architectural knowledge as a architectural view supports separation of concerns with regard to the various models in the system. We demonstrate our approach for a compliance meta-data view for process-driven SOA systems that records compliance requirements and rationales of an architecture.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Domain-specific architectures; D.2.11 [**Software Architectures**]: Service-oriented architecture (SOA)

General Terms

Design, Documentation, Legal Aspects

Keywords

Architectural Knowledge, Compliance, MDD

1. INTRODUCTION

Architectural Knowledge (AK) tends to evaporate as software systems evolve, with grave consequences for software development projects [14]. A number of approaches have been proposed to generically solve this problem, for example based on text templates for AK [23] or based on meta-models to describe the AK [24, 14]. While these approaches work well in general, in the daily business, AK capturing is considered as an afterthought or not at all [25]. Retrospective modeling of AK is often seen as a painful additional responsibility without many gains [25]. It is still unclear how to capture AK without introducing efforts that outweigh the benefits [23, 3].

In this paper, we present a feasibility study for the idea to provide an architectural view [10] that domain-specifically captures one specific kind of AK – that is relevant to a project – using a custom meta-model and attach it to the sources and models of the project via model-driven technology. While there is the downside to this approach that the project needs to design and implement this domain-specific AK view, whereas the generic approaches are ready to use out-of-the-box, there are also a number of potential benefits. Recording only domain-specific AK means that only relevant information is recorded. The effort to design and implement the AK view requires the project to investigate in depth which AK is really needed. This further strengthens the relevance of the recorded information. Finally, via the model-driven approach, we can produce useful output in addition to making the AK sharable and reusable. For example, some AK might be used to generate online and offline software documentations.

In this paper, we investigate on this approach using the case of compliance in service-oriented architectures. 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 formalize, and before designing the software often an interpretation of the laws and regulations by domain experts or judicial experts is needed. Hence, it is difficult to see from a given SOA design or implementation, how the architecture supports the compliance requirements. In other words, the rationale of the SOA design gets lost.

This case is particularly interesting because recording the rationale and the requirements is required for a different reason than only the AK sharing and reuse: In an audit, many companies must be able to show that they comply to the laws and regulations. Our idea is to record this compliance meta-data alongside the system design and implementation in a special domain-specific AK view. In this view, we provide the compliance requirements that led to the architectural design along with links to the architectural components (in this case mainly services and processes) that implement the compliance requirements (giving a rationale for the architectural configuration of these components). While this information might not provide all the information, e.g., a full-blown architectural decision model provides, it is still useful enough to understand the main rationale of the decisions behind the compliance design.

Our approach makes the assumption that model-driven development (MDD) [19] is used to implement the architectural views and to generate the running system from the view models. MDD helps us to keep the data in the AK view up-to-date and consistent with the project because otherwise the system itself would be generated with incomplete or wrong data, or the generation might even fail.

This paper is organized as follows: First, in Section 2 we provide some background on compliance in SOAs as the area in which our approach is applied. Next, in Section 3 we introduce our View-based Modeling Framework that is used to implement process-driven SOAs in a view-based fashion using MDD techniques. Besides illustrating an example process, we also present the view extension mechanisms used to implement the compliance meta-data view, which is presented in Section 4. Next, in Section 5 we demonstrate how compliance documentation can be generated from such meta-data using model-driven transformation. In Section 6 we compare to the related work and in Section 7 we conclude.

2. COMPLIANCE IN SERVICE-ORIENTED ARCHITECTURES

In this paper, we use a case from the area of business compliance in process-driven SOAs to illustrate our model-driven and domain-specific architectural knowledge view approach.

A *service* is in first place a distributed object that is accessible via the network and has certain characteristics: The service offers a public interface and is both platform- and protocol-independent. It is self-contained in the sense that it can interdependently be used, and no implementation details need to be known for using the service. Service-oriented Architecture (SOA) is the main architectural style for service-oriented computing. In this paper we focus on a particular kind of SOAs, which are process-driven. In a *process-driven SOA*, a process engine is used to orchestrate the services in order to implement business processes [8].

IT compliance means in general complying with laws and regulations applying to an IT system, such as the Basel II Accord [2], the International Financial Reporting Standards (IFRSs) [12], the Markets in Financial Instruments Directive (MiFID) [6], the Financial Security Law of France (LSF) [18], the Dutch Corporate Governance Code (Code-Tabaksblat) [21], and the Sarbanes-Oxley Act (SOX) [4]. These cover issues such as auditor independence, corporate governance, and enhanced financial disclosure. Laws and regulations are, however, just one example of compliance concerns that occur in process-driven SOAs. There are many other rules and constraints in a SOA that have similar characteristics. Some examples are service composition and deployment rules, service execution order rules, information exchange policies, security policies, qualitiy of service (QoS) rules, internal business rules, laws, and licenses.

Compliance concerns stemming from regulations or other compliance sources can be realized using a number of so-called *controls*. A control is any measure taken to assure a compliance requirement is met. For instance, an intrusion detection system, a firewall, or a business process realizing separation of duty requirements are all controls for ensuring systems security. As regulations such as SOX are not very concrete on how to realize the controls, usually, the regulations are mapped to established *norms* and *standards* describing more concretely how to realize the controls for a regulation. For example, COBIT [11] is a standard framework that defines among others controls for ensuring system security like the examples named before. A *risk assessment* is necessary to understand for instance the possible impacts of missing or failing controls. Controls can be realized in a number of different ways, including manual controls, reports, or automated controls.

3. VIEW-BASED MODELING FRAMEWORK

In this section, we give an overview of the View-based Modeling Framework (VbMF) [22] that is used as a foundation for implementing our AK views. It is a model-driven infrastructure that can generate code for process-driven SOAs in a view-based fashion. After explaining the basics of VbMF, we explain the view extension mechanisms used to implement our domain-specific AK view for compliance. Having such view extension mechanisms in place is an important part of our solution for model-driven AK views: This way the AK view can extend or annotate the existing models that are used to generate the system.

A typical business process in a SOA embodies various tangled concerns, such as the control flow, data processing, service and process invocations, fault handling, event handling, human interactions, transactions, to name but a few. The entanglement of those concerns increases the complexity of process development and maintenance as the number of involved services and processes grow. In order to deal with this complexity, we use the notion of architectural views [10] to describe the various SOA concerns. In particular, a view is a representation of one particular concern of a process. We devise different view models for formalizing the concept of architectural view.

Figure 1 shows basic process concerns such as the control flow, service invocations, and data processing, in terms of the flow view, collaboration view, and information view model, respectively. In addition to these concerns, the human view (cf. [9]) captures human aspects of business processes, i.e., the participation of users in processes. Finally, the transaction view allows to define transactions within the control flow. All these view models are built up around a Core model shown in Figure 2. The Core model is intentionally developed for conceptually representing the essence of a business process and the services with which it interoperates. That is, the Core model covers three distinct concepts: the process, the relationships between process and the environment, i.e., the services, and the internal representation of the process, i.e., the process views. Process concerns described by the view models merely relate to these concepts in the sense that each concern involves either the process's interior or exterior, or both. In other words, the other view models derive and extend the foundational concepts provided in the Core model as shown in Figure 1. 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 [22].

3.1 Separation of Concerns

Our view-based approach is not limited to these concerns, but can be extended to cover various other concerns. For instance, human interactions, transactions, event handling have been realized as extensions [22, 9]. This view extension mechanism is also used

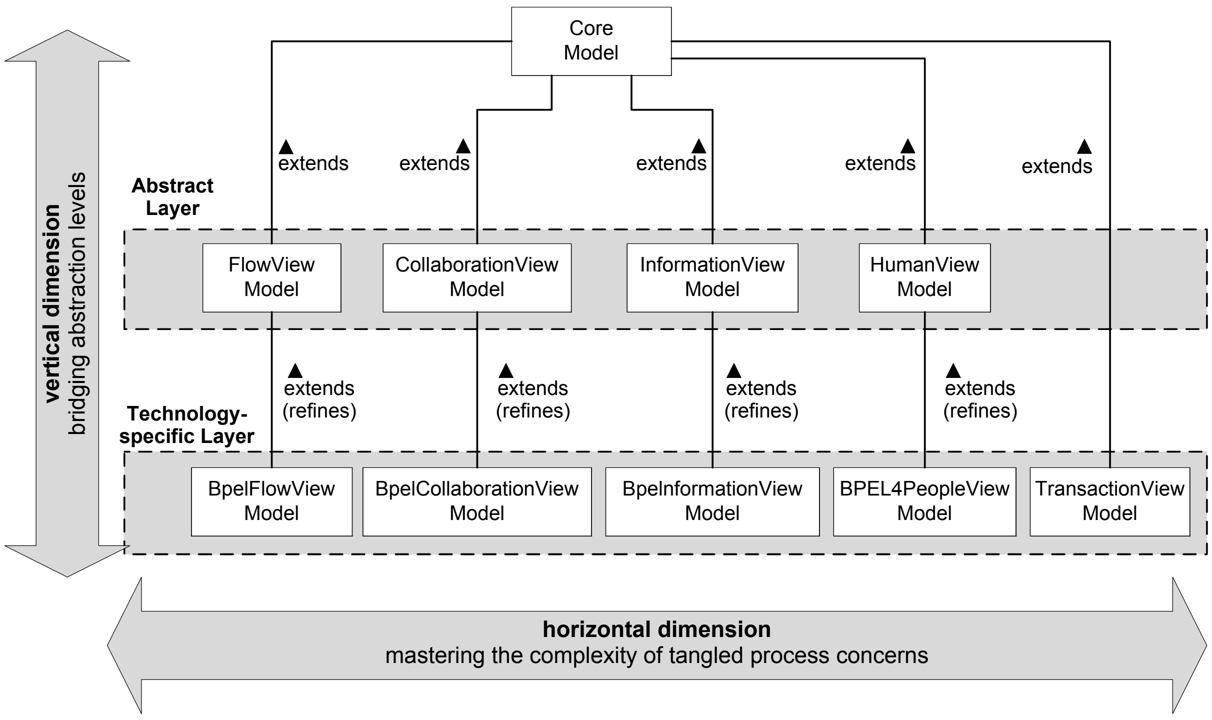


Figure 1: Layered Architecture of the View-based, Model-driven Approach

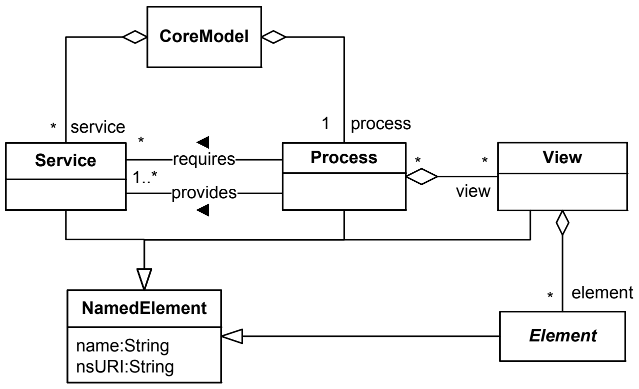


Figure 2: VbMF Core Model – Used for View Integration

later for introducing our domain-specific AK view.

A new concern can be integrated into our approach by using a corresponding *New-Concern*-View model that extends the basic concepts of the Core model and defines additional concepts of that concern. By adding new view models for additional process concerns, we can extend the view-based approach along the horizontal dimension, i.e., the dimension of process concerns, to deal with the complexity caused by the various tangled process concerns.

3.2 Abstraction Levels

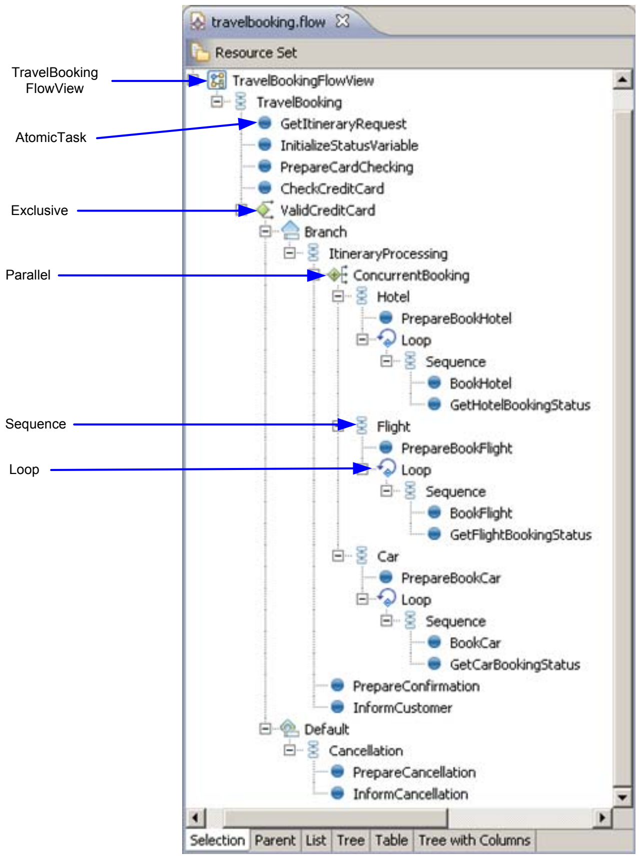
There are many stakeholders involved in process development at different levels of abstraction. For instance, business experts require high-level abstractions that offer domain or business concepts concerning their distinct knowledge, skills, and needs, while IT experts merely work with low-level, technology-specific descriptions.

The MDD paradigm [19] provides a potential solution to this problem by separating the platform-independent and platform-specific models. A platform-independent model is a model of a software system that does not depend on the specific technologies or platforms used to implement it while a platform-specific model links to particular technologies or platforms. Leveraging this advantage of the MDD paradigm, we devise 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. Then, the IT experts can refine or map these abstract concepts into platform- and technology-specific views.

The technology-specific layer contains the views that embody concrete information of technologies or platforms. On the one hand, a technology-specific view model can be directly derived from the Core model, such as the transaction view model shown in Figure 1. On the other hand, a technology-specific view model can also be an extension of an abstract one, for instance, the BPEL collaboration view model extends the collaboration view model, the BPELPeople view model extends the human view model, etc., by using the model refinement mechanism. By refining an abstract layer down to a technology-specific layer, our view-based approach helps bridging the abstraction levels along the vertical dimension, i.e., the dimension of abstraction, which is orthogonal to the horizontal dimension.

According to the specific needs and knowledge of the stakeholders, views can be combined to provide a richer view or a more thorough view of a certain process. For instance, IT experts may need to involve the process control flow along with service interactions which is only provided via an integration of the flow view with either the collaboration view or BPEL collaboration view.

Based on the aforementioned view model specifications, stakeholders can create different types of views for describing specific business processes. These process views can be instances of the concerns' view models, extension view models, or integrated view models (see Figure 1). They can be manipulated by the stakeholders to achieve a certain business goal, or adapt to new requirements in business environment or changes in technology and platform. Finally, we provide model-to-code transformations (or so-called code generations) that take these views as inputs and generate process implementations and deployment configurations. The resulting code and configurations, which may be augmented with hand-written code, can be deployed in process engines and application servers for execution.



```
travelbooking.flow
Resource Set
TravelBookingFlowView
  TravelBooking
    GetItineraryRequest
    InitializeStatusVariable
    PrepareCardChecking
    CheckCreditCard
    ValidCreditCard
      Branch
        ItineraryProcessing
          ConcurrentBooking
            Hotel
              PrepareBookHotel
              Loop
                Sequence
                  BookHotel
                  GetHotelBookingStatus
            Flight
              PrepareBookFlight
              Loop
                Sequence
                  BookFlight
                  GetFlightBookingStatus
            Car
              PrepareBookCar
              Loop
                Sequence
                  BookCar
                  GetCarBookingStatus
          PrepareConfirmation
          InformCustomer
      Default
        Cancellation
          PrepareCancellation
          InformCancellation
Selection | Parent | List | Tree | Table | Tree with Columns
```

Figure 3: Flow View: Travel Booking Example

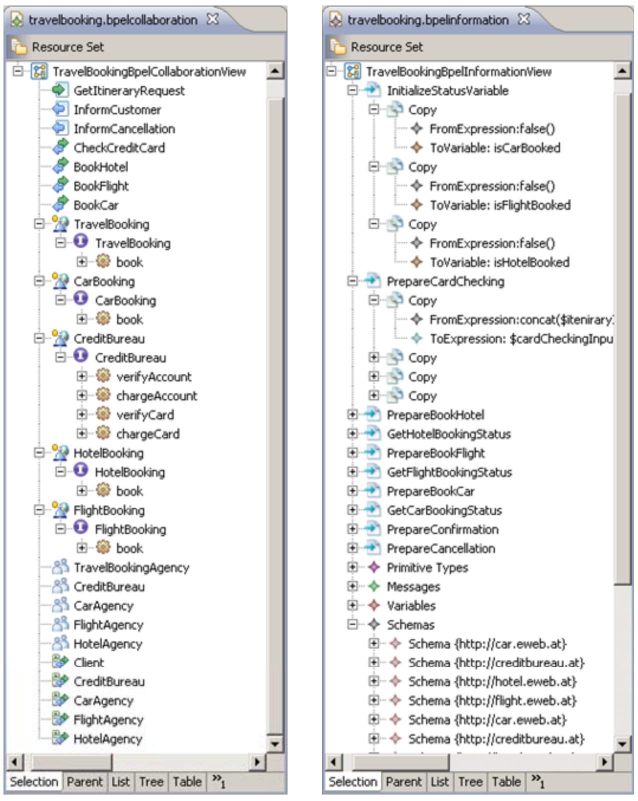
3.3 Integration of Views

Views can be integrated via integration points to produce a richer view or a more thorough view of the business process. We devise a name-based matching algorithm for realizing the view integration mechanism. This algorithm is simple, but effectively used at the view level (or model level) because from a modeler's point of view in reality, it makes sense, and is reasonable, to assign the same name to the modeling entities that pose the same functionality and semantics. Nonetheless, other view integration approaches such as those using class hierarchical structures or ontology-based structures are applicable in our approach with reasonable effort as well (see [22] for details).

3.4 Example: Travel Booking Process

Figure 3 shows an example of the flow view for a travel booking application (modeled using the VbMF Eclipse perspective). As can be seen the usual concepts of process modeling can be used in the flow view. Figure 4 shows two extensional views in the BPEL specific variant: the BPEL collaboration and information views. The collaboration view depicts information on the components (i.e., services) the process collaborates with and how collaboration is achieved. The information view depicts information on how data is passed in, into, and out of the process, as well as the business objects the process deals with.

The views are inter–related implicitly via the integration points from the Core view. Following the name-based matching convention we use the same names – e.g., for the services – in the different views.



```
travelbooking.bpelcollaboration
Resource Set
TravelBookingBpelCollaborationView
  GetItineraryRequest
  InformCustomer
  InformCancellation
  CheckCreditCard
  BookHotel
  BookFlight
  BookCar
  TravelBooking
    TravelBooking
      book
  CarBooking
    CarBooking
      book
  CreditBureau
    CreditBureau
      verifyAccount
      chargeAccount
      verifyCard
      chargeCard
  HotelBooking
    HotelBooking
      book
  FlightBooking
    FlightBooking
      book
  TravelBookingAgency
  CreditBureau
  CarAgency
  FlightAgency
  HotelAgency
  Client
  CreditBureau
  CarAgency
  FlightAgency
  HotelAgency
Selection | Parent | List | Tree | Table | »1
```

a) TravelBooking BpelCollaborationView

```
travelbooking.bpelinformation
Resource Set
TravelBookingBpelInformationView
  InitializeStatusVariable
    Copy
      FromExpression:false()
      ToVariable: isCarBooked
    Copy
      FromExpression:false()
      ToVariable: isFlightBooked
    Copy
      FromExpression:false()
      ToVariable: isHotelBooked
  PrepareCardChecking
    Copy
      FromExpression:concat($itenirary
      ToExpression: $cardCheckingInpu
    Copy
    Copy
    Copy
  PrepareBookHotel
  GetHotelBookingStatus
  PrepareBookFlight
  GetFlightBookingStatus
  PrepareBookCar
  GetCarBookingStatus
  PrepareConfirmation
  PrepareCancellation
  Primitive Types
  Messages
  Variables
  Schemas
    Schema {http://car.eweb.at}
    Schema {http://creditbureau.at}
    Schema {http://hotel.eweb.at}
    Schema {http://flight.eweb.at}
    Schema {http://car.eweb.at}
    Schema {http://creditbureau.at}
Selection | Parent | List | Tree | Table | »1
```

b) TravelBooking BpelInformationView

Figure 4: BPEL Collaboration and Information View: Travel Booking Example

4. DESIGN OF THE COMPLIANCE META-DATA VIEW

In this section, we illustrate the design of a compliance meta-data view. We propose a Compliance Meta-data view that allows for annotation of SOA elements with different compliance requirements. This is done by annotating process-driven model instances with the compliance meta-data. In particular, we elaborate on how to express compliance requirements originating from some compliance documents for process-driven SOAs using the VbMF. That is, we want to implement a compliance *control* for, e.g., a compliance regulation, standard, or norm, for a process or service.

The Compliance Meta-data view provides domain-specific 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

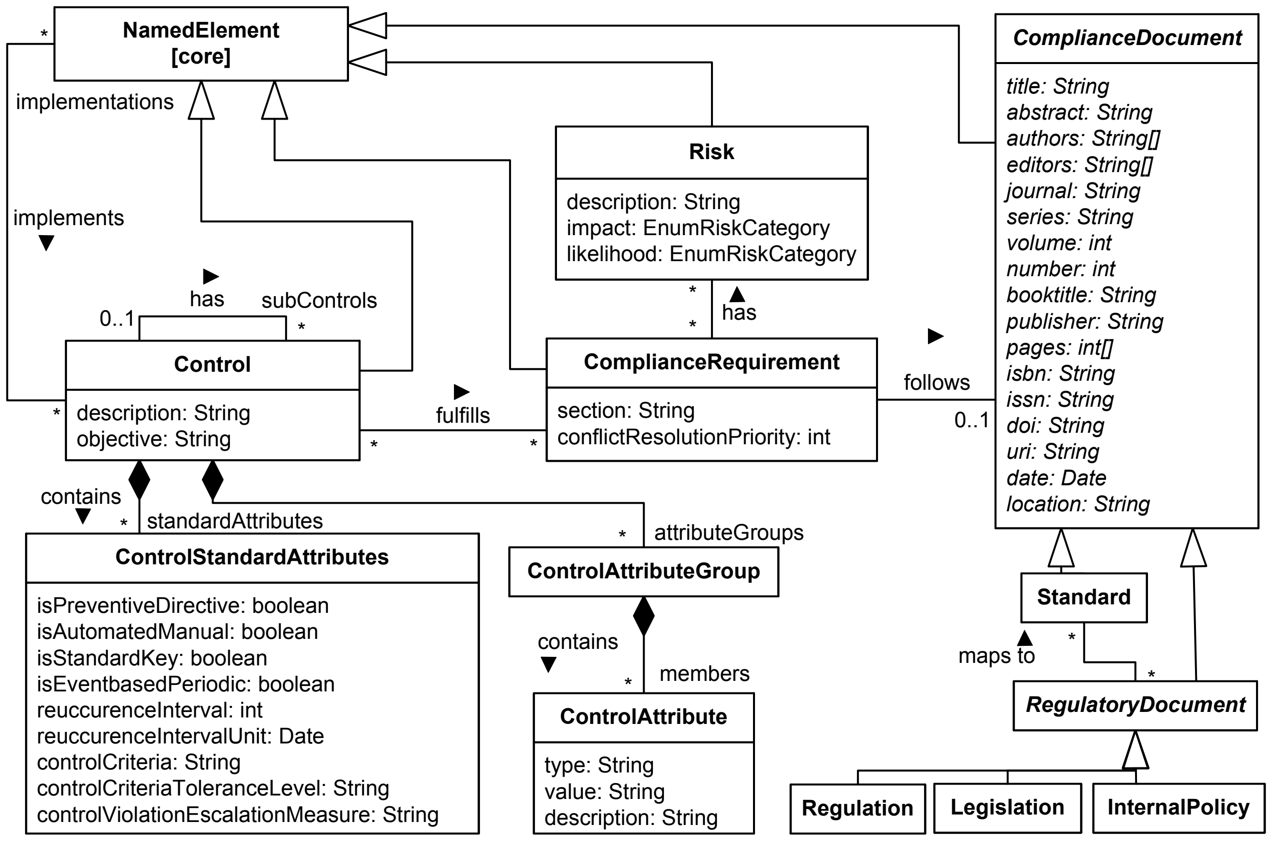


Figure 5: The Compliance Meta-data Model

in a certain architectural configurations. But in addition, the Compliance Meta-data view has other useful aspects for the project: From it, we can automatically generate compliance documentation for offline use (i.e., PDF[1] documents) and for online use (see also Section 5). 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, or a business concern. 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 5 shows our proposed Compliance Meta-data view. 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 views of the VbMF). A `Control` can have `subControls`. This way compliance controls can be grouped and combined. `Controls` fulfill `Compliance-Requirements` that relate to `ComplianceDocuments` such as a `Regulation`, `Legislation`, or `InternalPolicy`. Such `RegulatoryDocuments` can be mapped to `Standards` that represent another type of `ComplianceDocument`. Different categories of `ComplianceRequirements` are shown in Table 1.

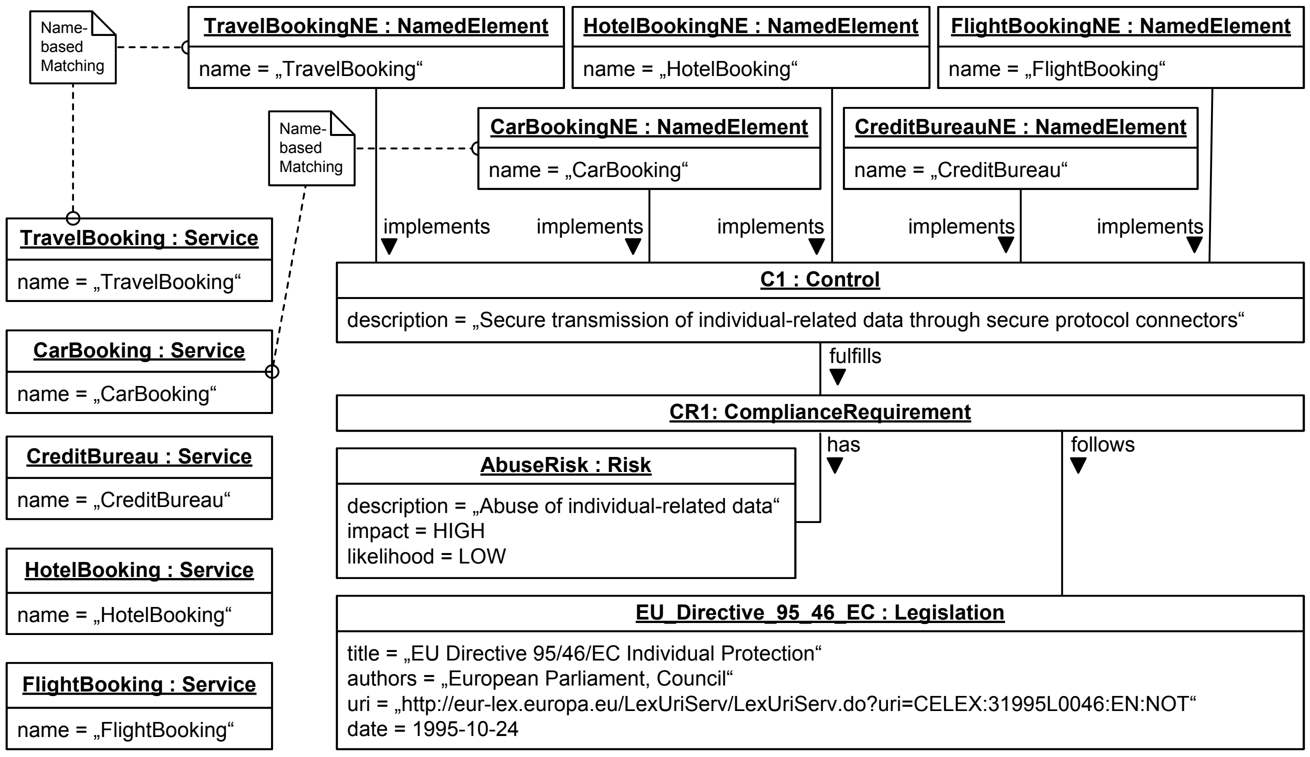
[1]Portable Document Format [13]

Compliance Concern	Description
Control flow	Order and execution of process elements
Locative	Execution location of processes and process elements (e.g., a certain host, within a company, or a country)
Information	Syntax and semantics of used or produced information
Resource	Involvement of resources in processes (e.g., human resources, CPU cycles, memory, disk-space)
Temporal	Temporal constraints on process execution (e.g., deadlines, scheduled activities)

Table 1: Categories of Compliance Requirements

When there exists a compliance requirement, it usually comes with risks that arise from a violation of it. `Risks` have dimensions such as `likelihood` or `impact`. In this work we provide basic support for specifying such dimensions using linear comparable constants. Of course, these can be refined with more elaborative modeling elements that allow for non-trivial functions and the use of parameters, e.g., for probability density functions.

One important aspect when implementing compliance for a SOA is that we want to persist the relationship of a compliance requirement as derived from, e.g., a certain regulation or standard with the respective annotated SOA element. This allows for the identification and resolution of SOA elements, compliance controls, regulations, risks and compliance documents, e.g., in the case of a root-cause analysis of compliance violations.



Name-based Matching
TravelBookingNE : NamedElement
name = „TravelBooking"
HotelBookingNE : NamedElement
name = „HotelBooking"
FlightBookingNE : NamedElement
name = „FlightBooking"
Name-based Matching
CarBookingNE : NamedElement
name = „CarBooking"
CreditBureauNE : NamedElement
name = „CreditBureau"
TravelBooking : Service
name = „TravelBooking"
CarBooking : Service
name = „CarBooking"
CreditBureau : Service
name = „CreditBureau"
HotelBooking : Service
name = „HotelBooking"
FlightBooking : Service
name = „FlightBooking"
implements
implements
implements
implements
implements
C1 : Control
description = „Secure transmission of individual-related data through secure protocol connectors"
fulfills
CR1: ComplianceRequirement
has
follows
AbuseRisk : Risk
description = „Abuse of individual-related data"
impact = HIGH
likelihood = LOW
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

Figure 6: Compliance Meta-data: Travel Booking Example

For documentation purposes and for the implementation of compliance controls the `ControlStandard-Attributes` 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 `ControlAttributeGroups`.

Figure 6 shows an example for compliance meta-data that contains a directive from the European Union on the protection of individuals with regard to the processing of personal data. The example extends the travel booking application example presented already in Figures 3 and 4. In particular, the compliance control is implemented by the services of the travel booking process. Hence, the views in Figures 3 and 4 provide the architectural configuration of the processes and services, and Figure 6 provides the compliance-related rationale for the design of this configuration.

The `C1` compliance control instance for a secure transmission of personal data annotates the `TravelBooking` service of the process. The fulfilled requirement `CR1` follows the legislative document and is associated with an `AbuseRisk`. Via the name-based matching convention (see Section 3.3) the SOA elements are annotated in the compliance meta-data view: on the left of Figure 6 the various services of the travel booking process are displayed next to the object instances of the compliance meta-data view. The compliance control `C1` associates various `NamedElements` that hold the same name as the corresponding services from the BpelCollaborationView as shown in Figure 4).

With the proposed compliance 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 [5] and is implemented by the TravelBooking service.* within the VbMF.

This 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. 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.

5. GENERATING COMPLIANCE DOCUMENTATION

The compliance meta-data not only serves for specifying the architectural knowledge of a process-driven SOA as described in the previous section but also can be used for reporting and documentation purposes. In particular, it can be used for *generating* documentations. Such documentations visualize compliance relevant information for various stakeholders, such as executive managers and auditors, and therefore, help them to quickly gain an overview of a thorough view. Hyperlinks to other documentation pages allow the user to navigate to related information or to request more specific details. In Figure 7 a model–to–code Xpand [20] transformation template is shown that generates HTML[2] code for online documentation. The generated website displays a matrix of controls from a compliance meta-data view instance that are correlated against risks.

[2]HyperText Markup Language [1]

```
<h2>Risk-Control Correlation Matrix</h2>
<table>
  <tr>
    <th>Risks/Controls</th>
«FOREACH cv.control AS c»
    <th>
      <a href="«cv.processName+
              "_C_"+c.uuid+".html"»">«c.name»</a>
    </th>
«ENDFOREACH»
  </tr>
«FOREACH cv.risk AS r»
  <tr>
    <th>
      <a href="«cv.processName+
              "_R_"+r.uuid+".html"»">«r.name»</a>
    </th>
  «FOREACH cv.control AS c»
    <td>
      «IF (c.requirements.risks.contains(r))»X«ENDIF»
    </td>
  «ENDFOREACH»
  </tr>
«ENDFOREACH»
</table>
```

Figure 7: A Model-driven Transformation Template for Generating Compliance Documentation

Other generated documentation of the compliance meta-data focuses on, e.g., the relation of compliance requirements and compliance documents, such as standards or legislative documents. Also, the *coverage* of SOA elements in regard to compliance aspects with their relation to compliance documents can be visualized and highlighted. Thus, while the generation of process code may already consider the meta-data during transformation (e.g., in order to make sure that a secure protocol is used for services that are annotated accordingly), documentation that describes domain-specific AK can be automatically created and updated. The former – i.e., the use of domain-specific AK during code-generation, e.g., for ensuring the security of the system as in our example – is a clear incentive for a developer to specify and provide the AK for the SOA. The latter – i.e., the model-driven generation of documentation – comes with the advantage of keeping the domain-specific AK up-to-date.

6. RELATED WORK

Much work on better support for codifying the AK has been done in the area of architectural decision modeling. Jansen and Bosch see software architecture as being composed of a set of design decisions [14]. 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. [24]. Tyree and Akermann proposed a highly detailed, generic template for architectural decision capturing [23].

Question, Options, and Criteria (QOC) diagrams [17] raise a design question, which points to the available solution options, and decision criteria are associated with the options. This way decisions can be modeled as such. Kruchten et al. 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 [16]. Falessi et al. present the Decision, Goal, and Alternatives framework to capture design decisions [7].

Recently, Kruchten et al. extended these ideas with the notion of an explicit decision view [15] – akin to the basic view-based concepts in our approach.

In contrast to our work, the related work on architectural decision modeling focuses on *generic* knowledge capturing. In contrast, 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 collaboration view, 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. In contrast, this is a focus of our work.

7. LESSONS LEARNED AND CONCLUSIONS

Our feasibility study showed that it is feasible in a model-driven project to add an additional model-driven view that adds AK meta-data with reasonable effort. We were also able 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. However, compared to the related work, that generically documents the AK, additional effort is needed. This is not always a liability as it might help a project to better understand which kinds of AK documentations are really required.

The model-driven approach helps to keep the data in the AK view up-to-date and *consistent* with the project because otherwise the system itself would be generated with incomplete or wrong data, or the generation might even fail. Hence, it makes sense to connect the data to be recorded in the AK view with other meta-data that needs to be recorded in the project anyway, so that there is an additional incentive for developers to record the AK. In our case, we could demonstrate an area where this is feasible: compliance in service-oriented systems. A lacking or missing compliance documentation can have severe legal consequences, which is a great incentive to record it correctly. Of course, finding such an area in which meta-data needs to be recorded that is relevant to the project and can be linked to AK for a given project can be hard. But our general approach can also be applied for custom AK without such additional incentives.

There is the danger in our approach that only specific AK – linked to a domain specific area like compliance – is recorded and other AK still gets lost. It is the responsibility of a project to make sure that all relevant AK for understanding an architecture gets recorded.

We can conclude that it is possible and useful to add a domain-specific AK view to a model-driven project – with reasonable effort. If extra incentives can be found, such as generating a documentation or documenting compliance, they should be used to motivate developers to keep the information in the AK view up-to-date and consistent with the system.

Our approach assumes a model-driven approach is used for the system. 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.

Acknowledgments

This work was supported by the European Union FP7 project COMPAS, grant no. 215175.

8. REFERENCES

[1] M. Baker, M. Ishikawa, S. Matsui, P. Stark, T. Wugofski, T. Yamakami, and S. McCarron. XHTML^{TM} basic 1.1. W3C recommendation, W3C, July 2008. [accessed in March 2010].

[2] Bank for International Settlements. Basel II: International Convergence of Capital Measurement and Capital Standards: A Revised Framework - Comprehensive Version. `http://www.bis.org/publ/bcbsca.htm`, June 2006. [accessed in March 2010].

[3] R. Capilla, F. Nava, and C. Carrillo. Effort estimation in capturing architectural knowledge. In *ASE*, pages 208–217. IEEE, 2008.

[4] Congress of the United States. Public Company Accounting Reform and Investor Protection Act (Sarbanes-Oxley Act), Pub.L. 107-204, 116 Stat. 745. `http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/content-detail.html`, July 2002. [accessed in March 2010].

[5] European Parliament and Council. Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:NOT`, October 1995. [accessed in March 2010].

[6] European Parliament and Council. Directive 2004/39/EC on markets in financial instruments. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:02004L0039-20060428:EN:NOT`, April 2004. [accessed in March 2010].

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

[8] C. Hentrich and U. Zdun. Patterns for process-oriented integration in service-oriented architectures. In *Proceedings of 11th European Conference on Pattern Languages of Programs (EuroPLoP 2006)*, Irsee, Germany, July 2006.

[9] T. Holmes, H. Tran, U. Zdun, and S. Dustdar. Modeling human aspects of business processes - a view-based, model-driven approach. In I. Schieferdecker and A. Hartman, editors, *ECMDA-FA*, volume 5095 of *Lecture Notes in Computer Science*, pages 246–261. Springer, 2008.

[10] IEEE. Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE-std-1471-2000, IEEE, 2000.

[11] Information Systems Audit and Control Association. Control Objectives for Information and Related Technology (CobiT). `http://www.isaca.org/cobit`, 1996. [accessed in March 2010].

[12] International Accounting Standards Committee (IASC) Foundation. International Financial Reporting Standards. `http://www.iasb.org/IFRSs/IFRS.htm`. [accessed in March 2010].

[13] International Organization for Standardization. ISO 32000-1:2008 document management – portable document format – part 1: Pdf 1.7. `http://www.iso.org/iso/catalogue_detail.htm?csnumber=51502`, July 2008. [accessed in March 2010].

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

[15] P. Kruchten, R. Capilla, and J. C. Duenas. The decision view's role in software architecture practice. *IEEE Software*, 26:36–42, 2009.

[16] P. Kruchten, P. Lago, and H. Vliet. Building up and reasoning about architectural knowledge. In C. Hofmeister, editor, *QoSA 2006 (Vol. LNCS 4214)*, pages 43–58, 2006.

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

[18] Ministre de l'économie, des finances et de l'industrie. loi de sécurité financière. `http://www.senat.fr/leg/pjl02-166.html`, February 2003. [accessed in March 2010].

[19] T. Stahl and M. Völter. *Model-Driven Software Development*. John Wiley & Sons, 2006.

[20] The Eclipse Foundation. Xpand. `http://www.eclipse.org/modeling/m2t/?project=xpand`. [accessed in March 2010].

[21] The Netherlands Corporate Governance Committee. The Dutch corporate governance code. `http://www.commissiecorporategovernance.nl/page/downloads/CODE%20DEF%20ENGELS%20COMPLEET%20II.pdf`, December 2003. [accessed in March 2010].

[22] H. Tran, U. Zdun, and S. Dustdar. View-based and model-driven approach for reducing the development complexity in process-driven SOA. In W. Abramowicz and L. A. Maciaszek, editors, *BPSC*, volume 116 of *LNI*, pages 105–124. GI, 2007.

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

[24] O. Zimmermann, T. Gschwind, J. Kuester, F. Leymann, and N. Schuster. Reusable architectural decision models for enterprise application development. In S. Overhage and C. Szyperski, editors, *Quality of Software Architecture (QoSA) 2007*, Lecture Notes in Computer Science, Boston, USA, July 2007. Springer-Verlag Berlin Heidelberg.

[25] O. Zimmermann, U. Zdun, T. Gschwind, and F. Leymann. Combining pattern languages and architectural decision models in a comprehensive and comprehensible design method. In *Working IEEE/IFIP Conference on Software Architecture (WICSA) 2008*, Vancouver, BC, Canada, February 2008.