Modeling Process-Driven SOAs – a View-Based Approach

Huy Tran, Ta'id Holmes, Uwe Zdun, Schahram Dustdar
Distributed Systems Group, Institute of Information Systems
Vienna University of Technology, Austria

ABSTRACT

This chapter introduces a view-based, model-driven approach for process-driven, service-oriented architectures. A typical business process consists of numerous tangled concerns, such as the process control flow, service invocations, fault handling, transactions, and so on. Our view-based approach separates these concerns into a number of tailored perspectives at different abstraction levels. On the one hand, the separation of process concerns helps reducing the complexity of process development by breaking a business process into appropriate architectural views. On the other hand, the separation of levels of abstraction offers appropriately adapted views to stakeholders, and therefore, helps quickly re-act to changes at the business level and at the technical level as well. Our approach is realized as a model-driven tool-chain for business process development.

INTRODUCTION

Service-oriented computing is an emerging paradigm that made an important shift from traditional tightly coupled to loosely coupled software development. Software components or software systems are exposed as services. Each service offers its functionality via a standard, platform-independent interface. Message exchange is the only way to communicate with a certain service.

The interoperable and platform independent nature of services underpins a novel approach to business process development by using processes running in process engines to invoke existing services from process activities (also called process tasks or steps). Hentrich and Zdun (2006) call this kind of architecture a process-driven, service-oriented architecture (SOA). In this approach, a typical business process consists of many activities, the control flow and the process data. Each activity corresponds to a communication task (e.g., a service invocation or an interaction with a human), or a data processing task. The control flow describes how these activities are ordered and coordinated to achieve the business goals. Being well considered in research and industry, this approach has led to a number of standardization efforts such as BPEL (IBM et al., 2003), XPDL (WfMC, 2005), BPMN (OMG, 2006), and so forth.

As the number of services or processes involved in a business process grows, the complexity of developing and maintaining the business processes also increases along with the number of invocations and data exchanges. Therefore, it is error-prone and time consuming for developers to work with large business processes that comprise numerous concerns. This problem occurs because business process descriptions integrate various concerns of the process, such as the process control flow, the data dependencies, the service invocations, fault handling, etc. In addition, this problem also occurs at different abstraction levels. For instance, the business process is relevant for different stakeholders:

Business experts require a high-level business-oriented understanding of the various process elements (e.g., the relations of processes and activities to business goals and organization units), whereas the technical experts require the technical details (e.g., deployment information or communication protocol details for service invocations).

Besides such complexity, business experts and technical experts alike have to deal with a constant need for change. On the one hand, process-driven SOA aims at supporting business agility. That is, the process models should enable a quicker reaction on business changes in the IT by manipulating business process models instead of code. On the other hand, the technical infrastructure, for instance, technologies, platforms, etc., constantly evolves.

One of the successful approaches to manage complexity is *separation of concerns* (Ghezzi et al., 1991). Process-driven SOAs use modularization as a specific realization of this principle. Services expose standard interfaces to processes and hide unnecessary details for using or reusing. This helps in reducing the complexity of process-driven SOA models. However, from the modelers' point of view, such abstraction is often not enough to cope with the complexity challenges explained above, because modularization only exhibits a single perspective of the system focusing on its (de-)composition. Other - more problem-oriented - perspectives, such as a business-oriented perspective or a technical perspective (used as an example above), are not exhibited to the modeler. In the field of software architecture, *architectural views* have been proposed as a solution to this problem. An *architectural view* is a representation of a system from the perspective of a related set of *concern*s (IEEE, 2000). The architectural view concept offers a separation of concerns that has the potential to resolve the complexity challenges in process-driven SOAs, because it offers more tailored perspectives on a system, but it has not yet been exploited in process modeling languages or tools.

We introduce in this chapter a view-based approach inspired by the concept of architectural views for modeling process-driven SOAs. Perspectives on business process models and service interactions - as the most important concerns in process-driven SOA - are used as central views in the view-based approach. This approach is extensible with all kinds of other views. In particular, the approach offers separated views in which each of them represents a certain part of the processes and services. Some important views are the collaboration view, the information view, the human interaction view and the control flow view. These views can be separately considered to get a better understanding of a specific concern, or they can be merged to produce a richer view or a thorough view of the processes and services.

Technically, the aforementioned concepts are realized using the model-driven software development (MDSD) paradigm (Völter and Stahl, 2006). We have chosen this approach to integrate the various view models into one model, and to automatically generate platform-specific or executable code in BPEL (IBM et al., 2003), WSDL (W3C, 2001) and XML Schema (W3C, 2001). In addition, MDSD is also used to separate the platform-specific views from the platform-neutral and integrated views, so that business experts do not have to deal with platform-specific details. The code generation process is driven by model transformations from relevant views into executable code.

This chapter starts by introducing some basic concepts and an overview of the view-based modeling framework. Then we give deeper insight into the framework which is followed by a discussion of view development mechanisms such as view extension, view integration and code generation mechanisms. A simple case study, namely, a Shopping process, is used to illustrate the realization of the modeling framework concepts. The chapter concludes with a discussion to summarize the main points and to broaden the presented topics with some outlooks.

OVERVIEW OF THE MODELING FRAMEWORK

In this section, we briefly introduce the View-based Modeling Framework (VbMF) which utilizes the MDSD paradigm. VbMF comprises modeling elements such as a meta-model, view models, and view instances (see Figure 1). In VbMF, a view (or a model) is a representation of a process from the perspective of related concerns. Each view instance comprises many relevant elements and relationships among these elements. The appearance of view elements and their relationships are precisely specified in a view model that the view must conform to. A view model, in turn, conforms to the meta-model at layer M2. We devise a simple meta-model, which is based on the meta-model of the Eclipse Modeling Framework (Eclipse EMF, 2006), as the cornerstone for the modeling framework. The framework view models are developed on top of that meta-model.

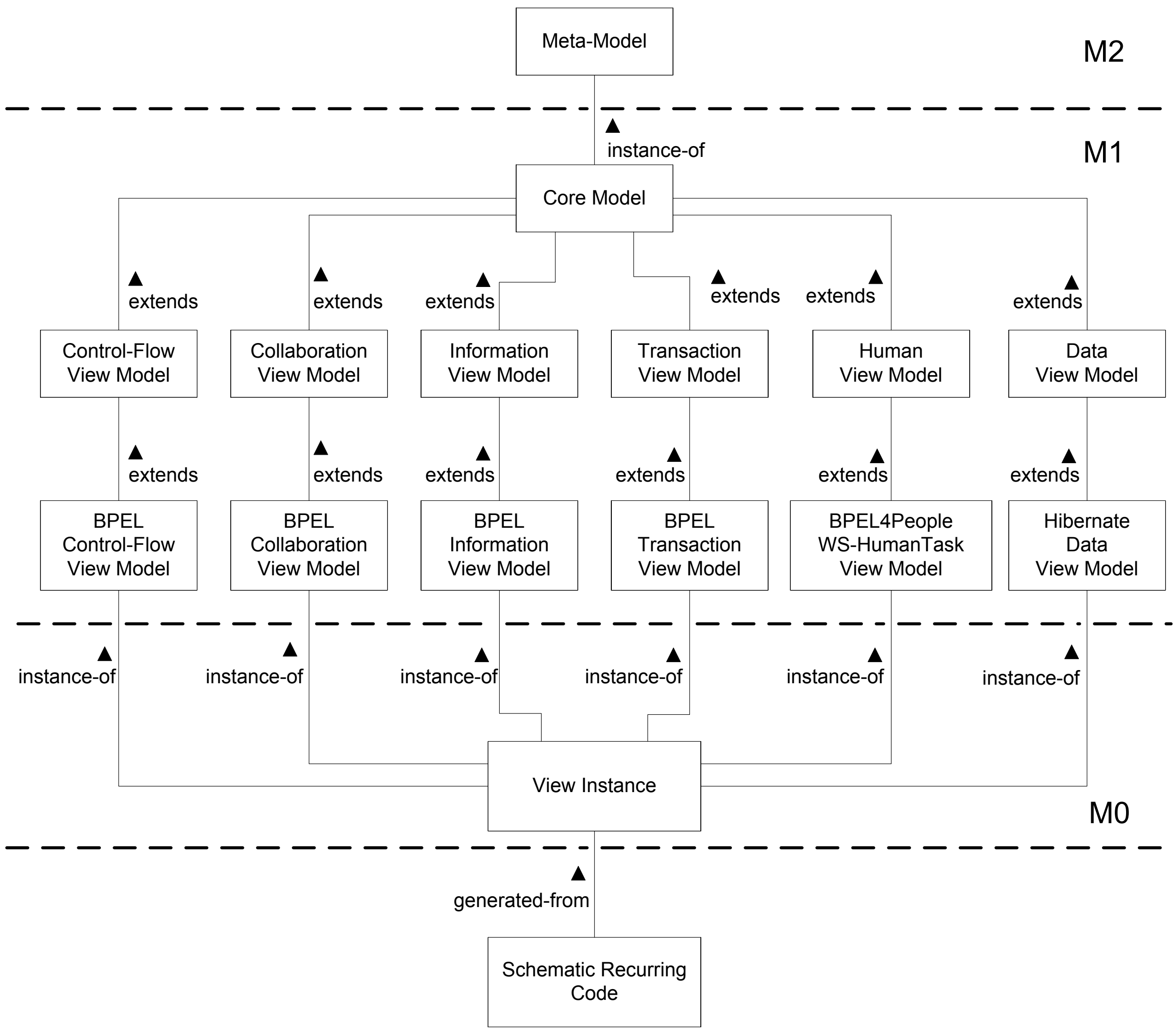

Figure 1 Layered architecture of View-based Modeling Framework

In our approach, we categorize distinct activities - in which the modeling elements are manipulated (see Figure 2):

- *Design* activities define new architectural view instances or new view models. This kind of activity includes *Extension activities* which create a new view model by adding more features to an existing view model.

- *Integration* activities are done by the View Integrator to combine view instances to produce a richer view or a thorough view of a business process.
- *Transformation* activities are performed by the Code Generator to generate executable code from one or many architectural views.
- *Interpretation* activities are used to extract relevant views from existing legacy business process code.

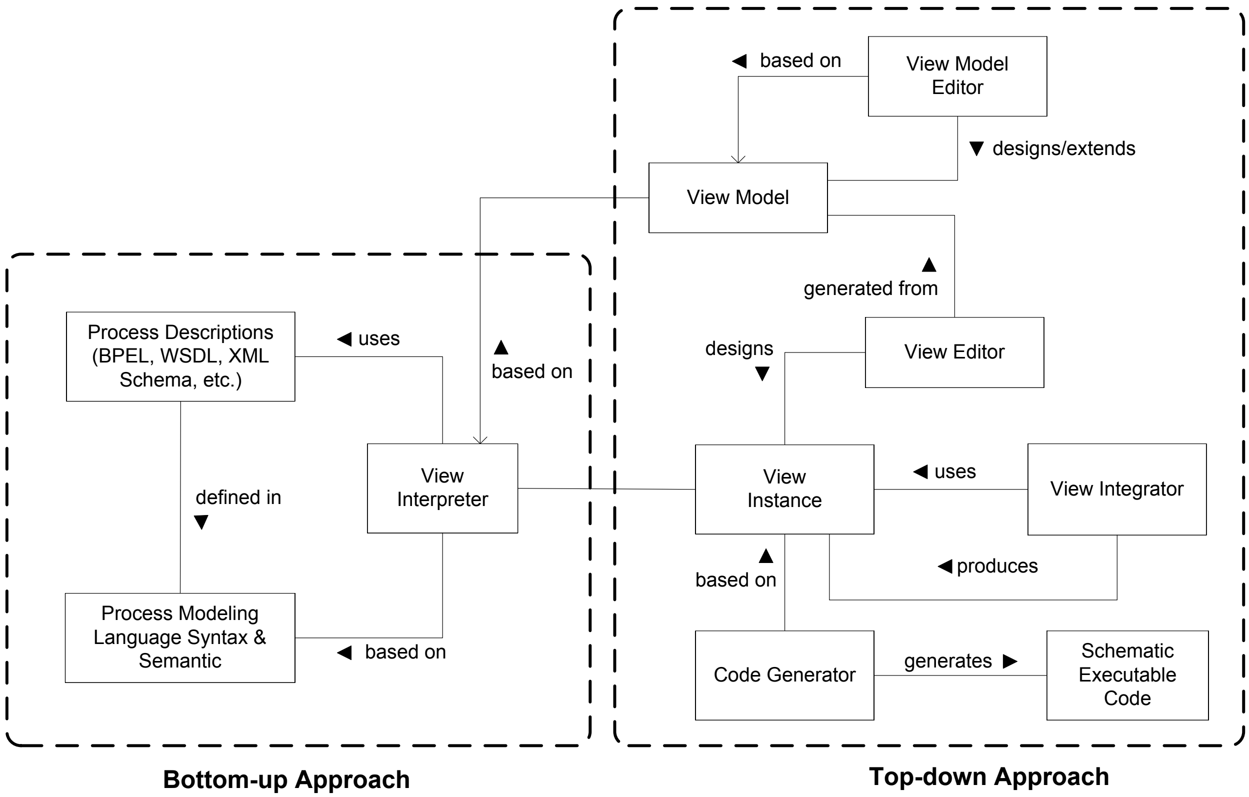

Figure 2 Top-down and bottom-up approach in View-based Modeling Framework

Before generating outputs, the View Integrator validates the conformity of the input views against corresponding view models. *Extension* and *Integration* are the most important activities used to extend our view-based model-driven framework toward various dimensions. Existing view models can be enhanced using the extension mechanisms or can be merged using the integration mechanisms as explained in the subsequent sections.

VIEW-BASED MODELING FRAMEWORK

A typical business process comprises various concerns that require support of modeling approaches. In this chapter we firstly examine basic process concerns such as the control flow, data handling and messaging, and collaboration (see Figure 3). However, the view-based modeling framework is not just bound to these concerns. The framework is fully open and extensible such that other concerns, for instance, transactions, fault and event handling, security, human interaction, and so on, can be plugged-in using the same approach. In the next sections, we present in detail the formalized representations of process concerns in terms of appropriate view models along with the discussion of the extensibility mechanisms *Extend* and *Integrate*.

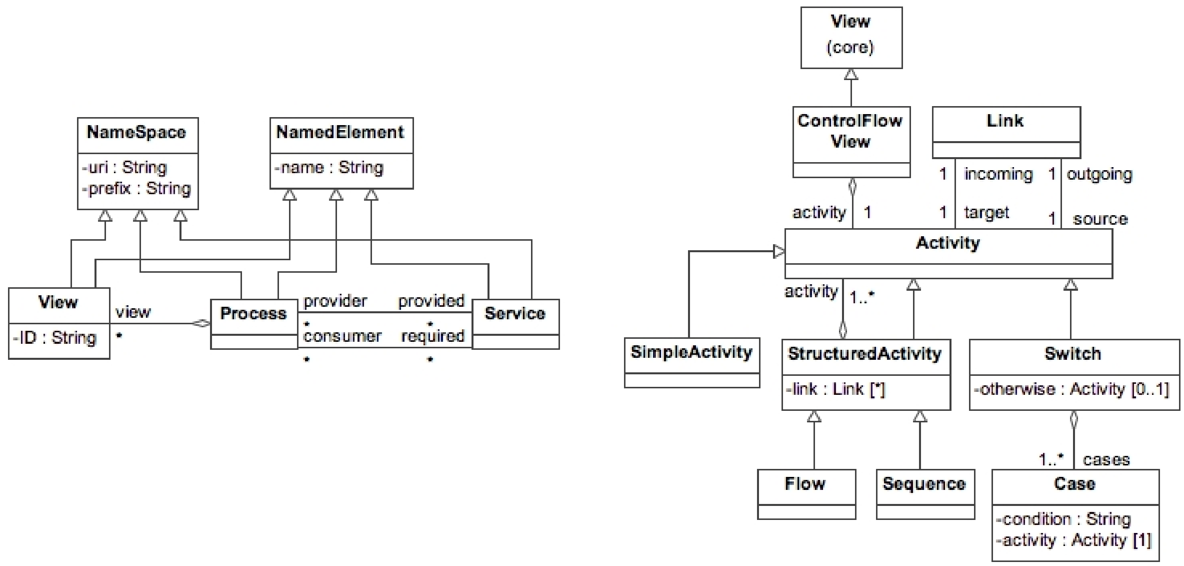

Figure 3 The Core model (left-hand side) and the Control-flow View model (right-hand side)

The Core model

Aiming at the openness and the extensibility, we devise a basic model, called the Core model, as a foundation for the other view models (see Figure 3). Each of the other view models is defined by extending the Core model. Therefore, the view models are independent of each other. The Core model is the place where the relationships among the view models are maintained. Hence, the relationships in the Core model are needed for view integrations.

The Core model provides a number of important abstract elements: *View*, *Process* and *Service*. Each of them can be extended further. At the heart of the Core model is the View element that captures the architectural view concept. Each specific view (i.e., each instance of the View element) represents one perspective on a particular Process. A Service specifies external functions that the Process provides or requires. A View acts as a container for modeling elements representing the objects which appear inside the Process. Different instances of each of these elements can be distinguished through the features of the common superclasses *NamedElement*, defining a name property, and *NameSpace,* defining an URI and prefix based namespace identifier.

The view models that represent concerns of a business process are mostly derived from the Core model. Therefore, these elements of the Core model are important extension points. The hierarchical structures in which those elements are roots can be used to define the integration points used to merge view models as mentioned in the description of the integration mechanisms below.

Control-flow View model

The control flow is one of the most important concerns of a SOA process. A Control-flow View comprises many activities and control structures. The activities are process tasks such as service invocations or data handling, while control structures describe the execution order of the activities to achieve a certain goal. Each Control-flow View is defined based on the Control-flow View model.

There are several approaches to modeling process control flows such as state-charts, block structures (IBM et al., 2003), activity diagrams (OMG, 2004), Petri-nets (Aalst et al., 2000), and so on. Despite of this diversity in control flow modeling, it is well accepted that existing modeling languages share five common basic patterns: *Sequence*, *Parallel Split, Synchronization*, *Exclusive Choice*, and *Simple Merge* (Aalst et al., 2003). Thus, we adopted these patterns as the building blocks of the Control-flow View model. Other, more advanced patterns can be added later by using extension mechanisms to augment the Control-flow View model. We define the Control-flow View model and semantics of the control structures with respect to these patterns (see Table 1).

Structure	Description
Sequence	An activity is only enabled after the completion of another activity in the same sequence structure. The sequence structure is therefore equivalent to the semantics of the *Sequence* pattern.
Flow	All activities of a flow structure are executed in parallel. The subsequent activity of the flow structure is only enabled after the completion of all activities in the flow structure. The semantics of the flow structure is equivalent to a control block starting with the *Parallel Split* pattern and ending by the *Synchronization* pattern.
Switch	Only one of many alternative paths of control inside a switch structure is enabled according to a condition value. After the active path finished, the process continues with the subsequent activity of the switch structure. The semantics of the switch structure is equivalent to a control block starting with the *Exclusive Choice* pattern and ending by the *Simple Merge* pattern.

Table 1: Semantics of basic control structures

The primary entity of the Control-flow View model is the *Activity* element (see Figure 3), which is the base class for other elements such as *Sequence, Flow*, and *Switch*. Another important entity in the Control-flow View model is the *SimpleActivity* class that represents a concrete action such as a service invocation, a data processing task, and so on. The actual description of each SimpleActivity is modeled in another specific view. For instance, a service invocation is described in a Collaboration View, while a data processing action is specified in an Information View. Each SimpleActivity is a placeholder or a reference to another activity, i.e., an interaction or a data processing task. Therefore, every SimpleActivity becomes an integration point that can be used to merge a Control-flow View with an Information View, or with a Collaboration View, respectively.

The *StructuredActivity* element is an abstract representation of a group of related activities. Some of these activities probably have logical correlations. For instance, a shipping activity must be subsequent to an activity receiving purchase orders. The *Link* element is used in such scenarios.

Collaboration View model

A business process is often developed by composing the functionality provided by various parties such as services or other processes. Other partners, in turn, might use the process. All business functions required or provided by the process are typically exposed in terms of standard interfaces (e.g., WSDL portTypes). We captured these concepts in the Core model by the relationships between the two elements Process and Service. The Collaboration View model (see Figure 4) extends the Core model to represent the interactions between the business process and its partners.

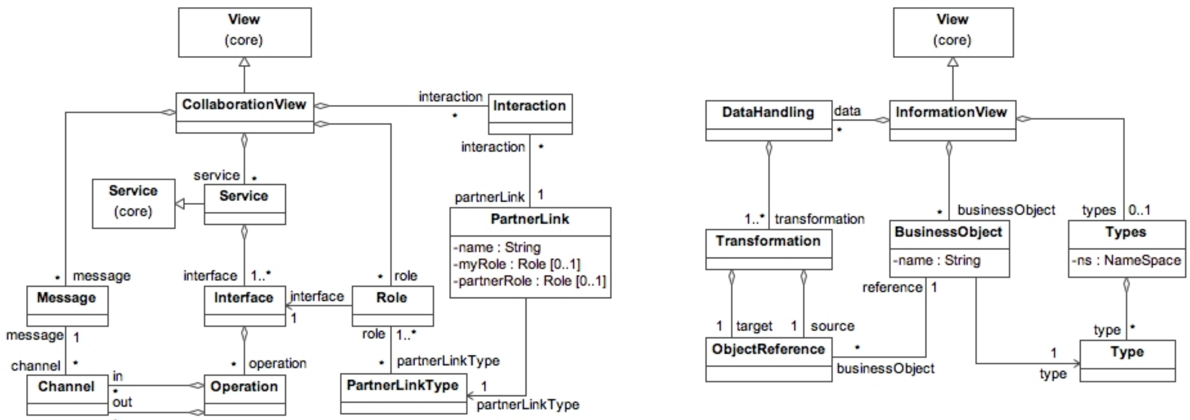

Figure 4 The Collaboration View model (left-hand side) and the Information View model (right-hand side)

In the Collaboration View model, the *Service* element from the Core model is extended by a tailored and specific *Service* element that exposes a number of *Interfaces*. Each Interface provides some *Operations*. An Operation represents an action that might need some inputs and produces some outputs via correspondent *Channels*. The details of each data element are not defined in the Collaboration View but in the Information View. A Channel only holds a reference to a *Message* entity. Therefore, each Message becomes an integration point that can be used to combine a specific Collaboration View with a corresponding Information View.

The ability and the responsibility of an interaction partner are modeled by the *Role* element. Every partner, who provides the relevant interface associated with a particular role, can play that role. These concepts are captured by using the *PartnerLink* and the *PartnerLinkType* elements and their relationships with the Role element. An interaction between the process and one of its partners is represented by the *Interaction* element that associates with a particular PartnerLink.

Information View model

The third basic concern we consider in the context of this chapter is information. This concern is formalized by the Information View model (see Figure 4). This view model involves the representation of data object flows inside the process and message objects traveling back and forth between the process and the external world.

In the Information View model, the *BusinessObject* element, which has a generic type, namely, *Type*, is the abstraction of any piece of information, for instance, a purchase order received from the customer or a request sent to a banking service to verify the customer's credit card, and so forth. Each Information View consists of a number of BusinessObjects. Messages exchanged between the process and its partners or data flowing inside the process might go through some *Transformations* that convert or extract existing data to form new pieces of data. The transformations are performed inside a *DataHandling* object. The source or the target of a certain transformation is an *ObjectReference* entity that holds a reference to a particular BusinessObject.

Human View model

So far we have examined different perspectives of a business process such as the control flow, the interaction with external process elements as described in the Collaboration View and the Information View. These essential views allow the specification of automated processes. If we are interested in processes that can be automated and that do not require human interaction, we may use these views for designing various processes. However, business processes often involve human participants. Certain process activities need appropriate human interactions. We name such process elements *Tasks*. Tasks, thus, are simple process activities that are accomplished by a person. Tasks may specify certain input values as well as a Task Description and may yield a result that can be represented using output values.

Besides the task as a special process element, the *Human View* as shown in Figure defines human roles and their relationships to the respective process and tasks. *Roles* are abstracting concrete users that may play certain roles. The Human View thus establishes a role-based abstraction. This role-based abstraction can be used for role-based access control (RBAC). RBAC, in general, is administered through roles and role hierarchies that mirror an enterprise's job positions and organizational structure. Users are assigned membership into roles consistent with a user's duties, competency, and responsibility.

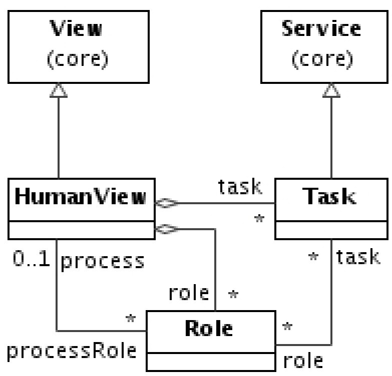

Figure 5 The Human View model

Examples for different roles are: Task Owner, Process Supervisor or Escalation Recipient. By binding, for instance, the role of a Process Supervisor to a process, RBAC can define that those users that are associated with this role may monitor the process execution. Similarly, the owner of a task may complete the task by sending results back to the process. He may however not follow up the process.
We can specify an activity as defined within a Control-flow View to be a human Task in the Human View that is bound to for instance an owner, the person who performs the task. Likewise, process stakeholders can be specified for the process by associating them with the human view.

Extension mechanisms

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

The aforementioned view models for the Control-flow, the Collaboration and the Information Views are the cornerstones to create abstract views. These abstract views aim at representing the high level, domain-related concepts, and therefore, they are useful for the business experts. According to the specific requirements on the granularity of the views, we can gradually refine these views toward more concrete, platform- or technology- specific views using the extension mechanisms.

A view refinement is performed by, firstly, choosing adequate extension points, and consequently, applying extension methods to create the resulting view. An extension point of a certain view is a view's element which is enhanced in another view by adding additional features (e.g., new element attributes, or new relationships with other elements) to form a new element in the corresponding view. Extension methods are modeling relationships such as generalization, extend, etc., that we can use to establish and maintain the relationships between an existing view and its extension. For instance, the Control-flow View, Collaboration View, and Information View models are mostly extensions of the Core model using the generalization relationship. We demonstrate the extensibility of the Collaboration View model by an enhanced view model, namely, the BPEL Collaboration View model (see Figure 6). Similar BPEL-specific view model extensions have also been developed for the Information View and the Control-flow View (omitted here for space reasons).

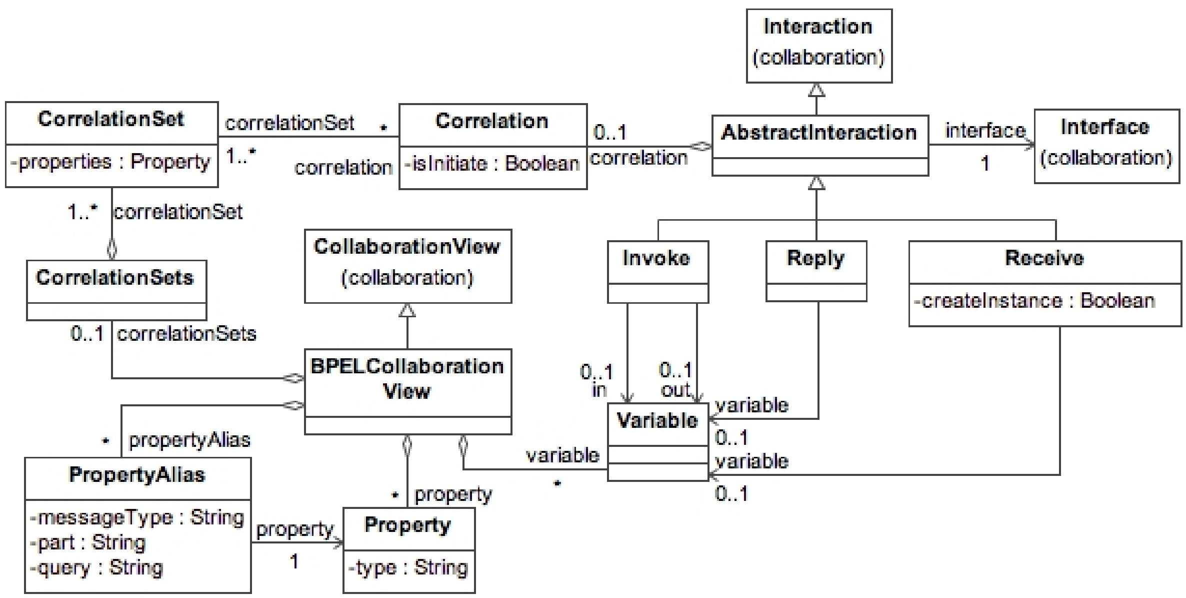

Figure 6 BPEL-specific extension of the Collaboration View

In the same way, more specific view models for other technologies can be derived. In addition, other business process concerns such as transactions, event handling, and so on, can be formalized by new adequate view models derived from the basic view model using the same approach as used above.

Integration mechanisms

In our approach, the Control-flow View - as the most important concern in process-driven SOA - is often used as the central view. Views can be integrated via integration points to provide a richer view or a thorough view of the business process. In the scope of this chapter, we utilize named-based matching mechanism for integrating views. This mechanism is effectively used at the view level (or model level) because from a modeler's point of view, it makes sense and is reasonable to give the same name to the modeling entities that pose the same functionality and semantics. However, other view integration approaches such as those using class hierarchical structures or ontology-based structures are applicable in the view-based modeling framework.

Model transformations

There are two basic types of model transformations: model-to-model and model-to-code. A model-to-model transformation maps a model conforming to a given meta-model to another kind of model conforming to another meta-model. Model-to-code, so-called code generation, produces executable code from a certain model. In the view-based modeling framework, the model transformations are mostly model-to-code that take as input one or many views and generate codes in executable languages, for instance, Java, BPEL/WSDL, and so on. In the literature, numerous code generation techniques are described, such as the combination of templates and filtering, the combination of template and meta-model, inline generation, or code weaving (Völter and Stahl, 2006). In our prototype, we used the combination of template and meta-model technique which is realized in the openArchitectureWare framework (oAW, 2002) to implement the model transformations. But any other of above-mentioned techniques could be utilized in this framework with reasonable modifications as well.

CASE STUDY

To demonstrate the realization of the aforementioned concepts, we explain a simple but realistic case study, namely, a Shopping process.

The Shopping process

The Shopping process is initiated when a certain customer issues a purchase order. The purchase order is retrieved via the *ReceiveOrder* activity. The process then contacts the Banking service to validate the credit card information through the *VerifyCreditCard* activity. The Banking service only needs some necessary information such as the owner's name, owner's address, card number, and expiry date. The process performs a preparation step, namely, *PrepareVerify,* which extracts such information from the purchase order. A preparation step is often executed before an interaction on the process takes place in order to arrange the needed input data for the interaction. After validating the customer's credit card, the control flow is divided into two branches according to the validation result. In case a negative confirmation is issued from the Bank service, e.g., because the credit card is invalid, the customer will receive an order cancellation notification along with an explaining message via the *CancelOrder* activity. Otherwise, a positive confirmation triggers the second control branch in which the process continues with two concurrent activities: *DoShipping* and *DoCharging*. The DoShipping activity gets delivery information from the purchase order and sends ordered products to the customer’s shipping address, while the DoCharging activity sends a request to the Banking service for the credit card's payment. Finally, the purchase invoice is prepared and sent back to the customer during the last step, *SendInvoice*. After that, the Shopping process successfully finishes.

Figure 7 shows the Shopping process developed using BPEL. VbMF can manage several important process concerns, for example, the control flow and service collaboration, data handling, fault and event handling, and transactions. For the demonstration purpose, in this chapter we only examine the control flow and service collaborations of the Shopping process. Therefore, in Figure 7, we present appropriate BPEL code and omit irrelevant parts.

```
<?xml version="1.0" encoding="UTF-8"?>
<bp:process name="Shopping"
  xmlns="http://www.shopping.com/"
  xmlns:shop="http://www.shopping.com/"
  xmlns:bank="http://www.banking.com/"
  xmlns:ship="http://www.shipping.com/"
  xmlns:bp="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <bp:partnerLinks>
    <bp:partnerLink name="Seller"
      partnerLinkType="shop:SellerPLT" myRole="Seller" />
    <bp:partnerLink name="Approver" partnerRole="Approver"
      partnerLinkType="shop:ApproverPLT" />
```

```
    <bp:partnerLink name="Payer" partnerRole="Payer"
      partnerLinkType="shop:PayerPLT" />
    <bp:partnerLink name="ShippingPartner" partnerRole="ShippingPartner"
      partnerLinkType="shop:ShippingPartnerPLT" />
  </bp:partnerLinks>

  <bp:variables>
    <bp:variable name="order_input" messageType="shop:PurchaseOrder" />
    <bp:variable name="order_output" messageType="shop:OrderResponse" />
    <bp:variable name="verify_input" messageType="bank:VerifyRequest" />
    <bp:variable name="verify_output" messageType="bank:VerifyResponse" />
    <bp:variable name="charge_input" messageType="bank:ChargeRequest" />
    <bp:variable name="charge_output" messageType="bank:ChargeResponse" />
    <bp:variable name="ship_input" messageType="ship:ShippingRequest" />
    <bp:variable name="ship_output" messageType="ship:ShippingResponse" />
  </bp:variables>

  <bp:sequence>
    <bp:receive name="ReceiveOrder"
      variable="order_input"
      partnerLink="Seller"
      portType="shop:Shopping"
      operation="doShopping"
      createInstance="yes" />
    <bp:assign name="PrepareVerify">
      <bp:copy>
          ...
      </bp:copy>
    </bp:assign>
    <bp:invoke name="VerifyCrediCard"
      inputVariable="verify_input"
      outputVariable="verify_output"
      partnerLink="Approver"
      portType="bank:CreditCard"
      operation="verifyCreditCard" />
    <bp:switch>
      <bp:case condition="condition">
        <bp:sequence>
          <bp:assign name="PrepareCancel">
            <bp:copy>
              ...
            </bp:copy>
          </bp:assign>
          <bp:reply name="CancelOrder"
            variable="order_output"
            partnerLink="Seller"
            portType="shop:Shopping"
            operation="doShopping" />
        </bp:sequence>
      </bp:case>
      <bp:otherwise>
        <bp:sequence>
          <bp:flow>
            <bp:sequence>
              <bp:assign name="PrepareShipping">
                <bp:copy>
                  ...
                </bp:copy>
              </bp:assign>
              <bp:invoke name="DoShipping"
                inputVariable="ship_input"
                outputVariable="ship_output"
                partnerLink="ShippingPartner"
```

```
                portType="ship:Shipping"
                operation="doShipping" />
            </bp:sequence>
            <bp:sequence>
              <bp:assign name="PrepareCharging">
                <bp:copy>
                  ...
              </bp:copy>
              </bp:assign>
              <bp:invoke name="DoCharging"
                inputVariable="charge_input"
                outputVariable="charge_output"
                partnerLink="Payer"
                portType="bank:CreditCard"
                operation="chargeCreditCard" />
            </bp:sequence>
          </bp:flow>
          <bp:assign name="PrepareInvoice">
            <bp:copy>
               ...
            </bp:copy>
          </bp:assign>

          <bp:reply name="SendInvoice"
            variable="order_output"
            partnerLink="Seller"
            portType="shop:Shopping"
            operation="doShopping" />
        </bp:sequence>
      </bp:otherwise>
    </bp:switch>
  </bp:sequence>
</bp:process>
```
Figure 7 Case study - the Shopping process developed using BPEL language

In the next paragraphs, we present an illustrative case study by the following steps. Firstly, the architectural views of the Shopping process are designed based on our view models and the sample extensions for BPEL constructs presented in the previous sections. These views are presented using the Eclipse Tree-based Editor (Eclipse EMF, 2006). Secondly, some views are integrated to produce a richer perspective. And finally, these views are used to generate executable code in WS-BPEL and WSDL that can be deployed into a BPEL engine.

View development

Figure 8 shows the Control-flow View instance of the Shopping process. There are no details of data exchanges or service communication in this view. Hence, the Control-flow View can be used by the stakeholders who need a high level of abstraction, for instance, the business experts or the domain analysts.

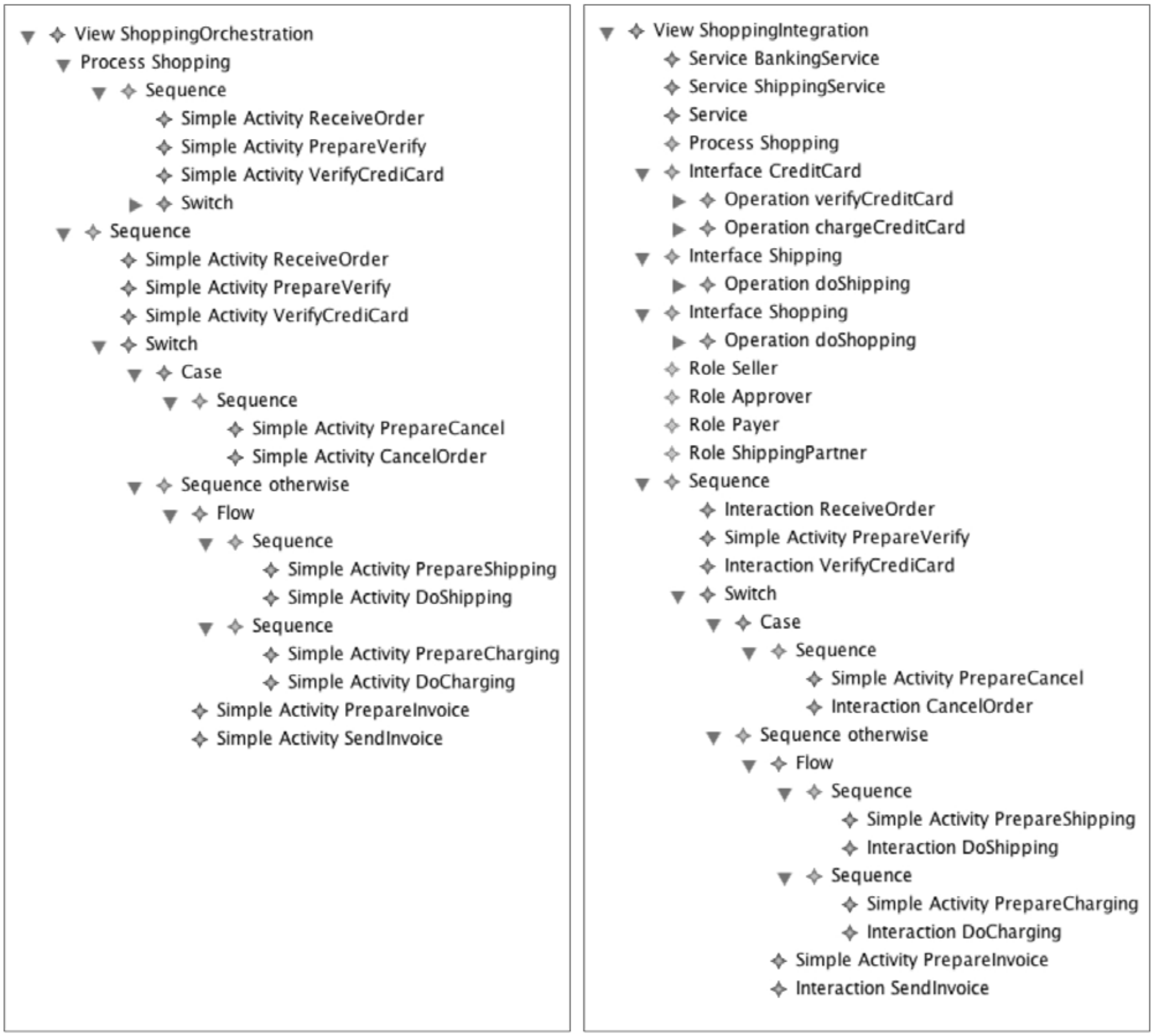

Figure 8 The Control-flow View (left-hand side) and an integrated view of the Shopping process – the result of integration the Control-flow View and the Collaboration View (right-hand side).

Moreover, using the extension view models (e.g., the BPEL-specific extension of the Collaboration View given in Figure 6), the technical experts or the IT developers can develop much richer views for a particular concern. In Figure 9, there are two models side by side in which one is the abstract collaboration model (i.e., the left-hand side view in Figure 9) and another one, which is at the right-hand side in Figure 9, is a view based on the BPEL Collaboration view model.

View integration

The views also can be integrated to produce new richer views of the Shopping process. At the right-hand side of Figure 9, we present an integrated view which is the result of the combination of the Control-flow View and the Collaboration View of the Shopping process. The SimpleActivity entities in the Control-flow View define the most important integration points with relevant Interaction entities in the Collaboration view. The output view consists of control structures based on the Control-flow View and additional collaboration-related entities such as Roles, Services, etc. Moreover, relevant activities of this view also comprise additional collaboration-specific attributes.

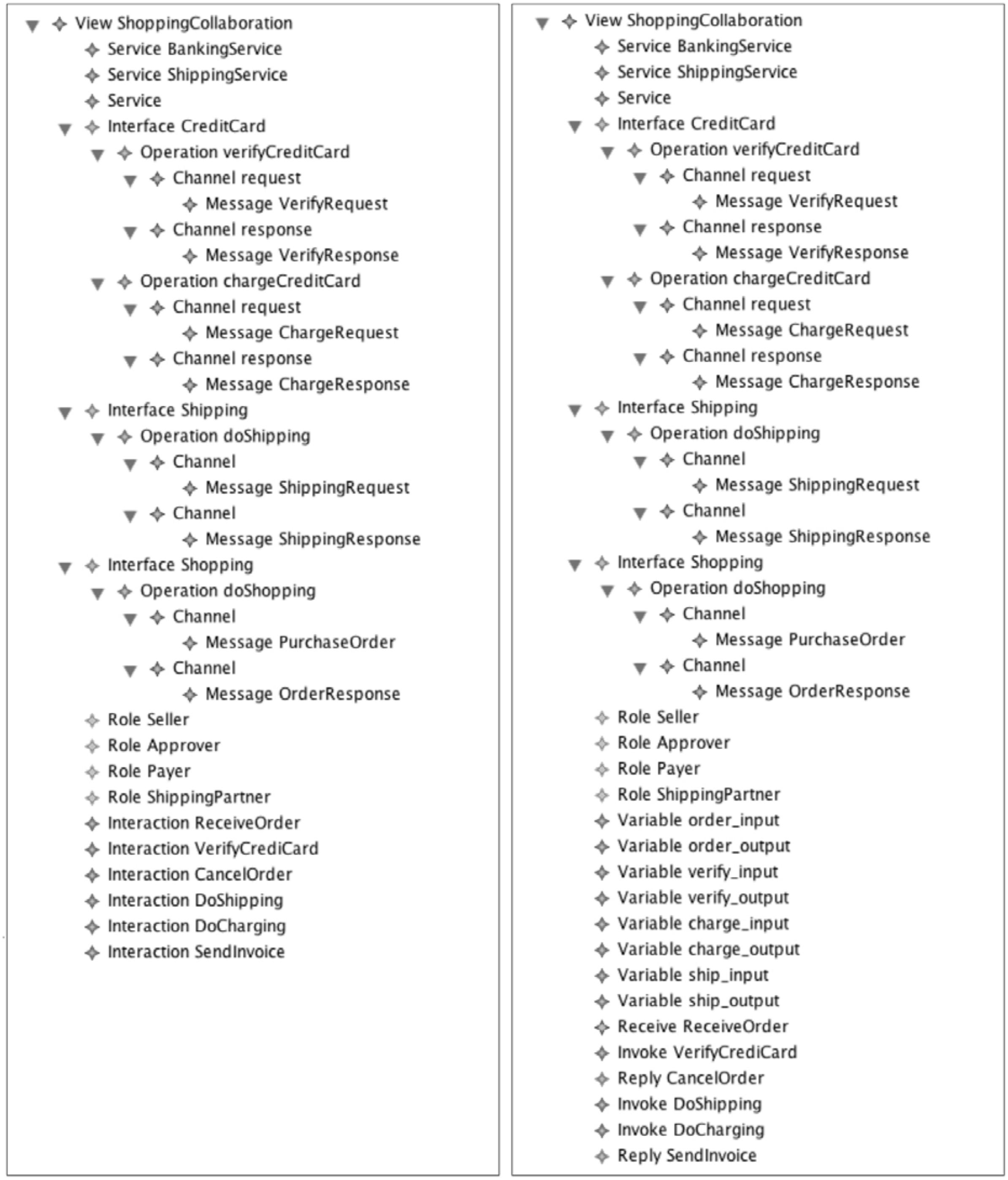

Figure 9 The Collaboration View (left-hand side) and the corresponding BPEL-specific extension view of the Collaboration View (right-hand side) of the Shopping process.

Code generation

After developing appropriate views for the Shopping process, we use illustrative template-based transformations to generate executable code for the process in BPEL and a service description in WSDL that represents the provided functions in terms of service interfaces. The modeling framework's models and Shopping process's models are EMF Ecore models (Eclipse EMF, 2006). We used the oAW's Xpand language (oAW, 2002) to define the code generation templates (see Figure 10).

```
#
#  Template for the main process
#
«DEFINE BPEL(core::View iv, core::View cv) FOR core::View»
«FILE process.name+".bpel"»
<?xml version="1.0" encoding="UTF-8"?>
<process name="«name»"
     «EXPAND Namespace FOR cv»
     xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            ......
     «EXPAND Control(iv, cv) FOR this»
</process>
«ENDFILE»
«ENDDEFINE»

#
# Template for the control structures
#
«DEFINE Control(core::View iv, core::View cv) FOR core::View»
     «LET getActivities(this) AS activities»
          «IF (activities != null && activities.size > 1)»
               <sequence>
               «EXPAND Activity(iv, cv) FOREACH activities»
               </sequence>
          «ELSEIF (activities != null && activities.size > 0)»
               «EXPAND Activity(iv, cv) FOREACH activities»
          «ENDIF»
     «ENDLET»
«ENDDEFINE»

#
# Template for generating code from the SimpleActivity of a Control-flow View
# Use named-based to integrate an appropriate SimpleActivity with an Interaction
# entity in a BPEL CollaborationView
#
«DEFINE Activity(core::View iv, core::View cv) FOR orchestration::SimpleActivity»
     «EXPAND SimpleActivity(iv, cv) FOR getActivityByName(name, iv, cv)»
«ENDDEFINE»

#
# Template for generating code from the Invoke activity
#
«DEFINE SimpleActivity(core::View iv,core::View cv) FOR bpelcollaboration::Invoke»
     <invoke name="«name»"
          «IF (in != null)»
               inputVariable="«getInput().name»"
          «ENDIF»
          «IF (out != null)»
               outputVariable="«getOutput().name»"
          «ENDIF»
          partnerLink="«partnerLink.name»"
          portType="«getRole().interface.name»"
          operation="«getOperation(getInterface(getRole())).name»"/>
«ENDDEFINE»
```

```
#
# Template for generating code from the Receive activity
#
«DEFINE SimpleActivity(core::View iv,core::View cv) FOR bpelcollaboration::Receive»
     <receive name="«name»"
          «IF (variable != null)»
               variable="«getVariable().name»"
          «ENDIF»
          «IF ( createInstance != null) »
               createInstance="«createInstance»"
          «ENDIF»
          partnerLink="«partnerLink.name»"
          portType="«getRole().interface.name»"
          operation="«getOperation(getInterface(getRole())).name»"/>
«ENDDEFINE»

#
# Template for generating code from the Reply activity
#
«DEFINE SimpleActivity(core::View iv,core::View cv) FOR bpelcollaboration::Reply»
     <reply name="«name»"
          «IF (variable != null)»
               variable="«getVariable().name»"
          «ENDIF»
          partnerLink="«partnerLink.name»"
          portType="«getRole().interface.name»"
          operation="«getOperation(getInterface(getRole())).name»"/>
«ENDDEFINE»
```

Figure 10 Templates in oAW's Xpand language for generating BPEL code from the Control-flow View and the BPEL-specific extension of the Collaboration View

We present a model transformation (aka code generation) snippet in oAW's Xpand language that generates executable code in BPEL language for activities such as Invoke, Receive and Reply using the BPEL-specific extension view given in Figure 3. The resulting executable code in BPEL and WSDL has been successfully deployed on the Active BPEL Engine (Active Endpoints, 2006).

CONCLUSION

Existing modeling approaches lack sufficient support to manage the complexity of developing large business processes with many different concerns because most of them consider the process model as a whole. We introduced in this chapter a view-based framework that precisely specifies various concerns of the process model and uses those models to capture a particular perspective of the business process. It not only helps to manage the development complexity by the separation of a business process's concerns, but also to cope with both business and technical changes using the separation of levels of abstraction. The proposed modeling framework can possibly be extended with other concerns of the business process such as security, event handling, etc., to cover all relevant concepts and process development technologies.

Acknowledgement. We would like to thank anonymous reviewers who provide useful feedback on an earlier draft of this chapter. This work was supported by the European Union FP7 project COMPAS, grant no. 215175.

SUGGESTED ADDITIONAL READING

There are several standardization efforts for process modeling languages, such as BPEL (IBM et al., 2003), BPMN (OMG, 2006), XPDL (WfMC, 2005), and so on. They can be categorized into different dimensions, for instance, textual and graphical languages, or abstract and executable languages. Most of these modeling languages consider the business process model as a whole, and therefore, do not support the separation of the process model's concerns. All these modeling languages can be integrated into the view-based modeling approach using extension models.

The concept of architectural views (or viewpoints) has potential of dealing with software development complexity, and therefore, is well-known in literature, for instance, the Open Distributed Processing Reference Model proposed in ISO (1998), or UML modeling language specified in UML (2003), to name a few. However, this concept has not been exploited in the field of business process development, and particularly, in process-driven SOA modeling. Axenath et al., (2005) present the Amfibia framework as an effort on formalizing different aspects of business process modeling, and propose an open framework to integrate various modeling formalisms through the interface concept. Akin to the approach presented in this chapter, Amfibia has the main idea of providing a modeling framework that does not depend on a particular existing formalism or methodology. The major contribution in Amfibia is to exploit dynamic interaction of those aspects. Therefore, the distinct point to VbMF is that in Amfibia the interaction of different "aspects" is only performed by event synchronization at run-time when the workflow management system executes the process. Using extension and integration mechanisms in VbMF, the integrity and consistency between models can be verified earlier at design time.

In this chapter, we also exploit the model-driven software development (MDSD) paradigm, which is widely used to separate platform-independent models from platform-specific models, to separate different levels of abstraction in order to provide appropriate adapted and tailored views to the stakeholders. Völter and Stahl (2006) provide a bigger, thorough picture about this emerging development paradigm in terms of the basic philosophy, methodology and techniques as well. Through this book, readers achieve helpful knowledge on basic terminologies such as meta-modeling, meta-meta-model, meta-model, model, platform-independent and platform-specific models, and modeling techniques such as model transformation, code generation as well.

Human interaction with SOAs have lately been formalized in The WS-BPEL Extension for People (BPEL4People) (Agrawal et al., 2007b). BPEL4People defines a *peopleActivity* as a new BPEL *extensionActivity* and thus realizes integration of human process activities into BPEL processes. BPEL4People is based on the WS-HumanTask specification that introduces formal definition of human tasks. Various roles for processes and tasks are defined in BPEL4People as well as WS-HumanTask that users can be assigned to for role-based access control.

REFERENCES

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

Aalst, W. van der, Hofstede, A. H. M. ter, Kiepuszewski, B., & Barros, A. P. (2003). Workflow patterns. *Distributed and Parallel Databases*, 14 (1), 5–51.

Active Endpoints (2006). ActiveBPEL Open Source Engine. http://www.active-endpoints.com.

Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., & Zeller, M. (2007a). Web Services Human Task (WS-HumanTask), Version 1.0. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf.

Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., & Zeller, M. (2007b). WS-BPEL Extension for People (BPEL4People), Version 1.0. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf.

Axenath, B., Kindler, E., & Rubin, V. (2005). An open and formalism independent meta-model for business processes. In *Proc. of the Workshop on Business Process Reference Models* (pp. 45–59).

Eclipse EMF. (2006). *Eclipse Modeling Framework*. http://www.eclipse.org/emf/.

Ferraiolo, D., Barkley, J., & Kuhn, D. R.. (1999). A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security* (TISSEC), 2(1), 34-64.

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (1991). *Fundamentals of Software Engineering*. Prentice Hall

Hentrich, C., & Zdun, U. (2006). Patterns for Process-oriented integration in Service-Oriented Architectures. In *Proc. of 11th European Conference on Pattern Languages of Programs (EuroPLoP'06)*. Irsee, Germany.

IBM, Systems, Microsoft, SAP AG, & Systems Siebel. (2003). *Business Process Execution Language for Web services*. ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

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

ISO. (1998). Open Distributed Processing Reference Model (IS 10746). http://isotc.iso.org/.

oAW. (2002) openArchitectureWare Project. http://www.openarchitectureware.org.

OMG. (2004). Unified Modelling Language 2.0 (UML). http://www.uml.org

OMG. (2006). Business Process Modeling Notation (BPMN). http://www.bpmn.org

Völter, M. & Stahl, T. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. Wiley.

W3C. (2001). Web Services Description Language 1.1. http://www.w3.org/TR/wsdl

W3C. (2001). XML Schema Part 1: Structures http://www.w3.org/TR/xmlschema-1/ and Part 2: Datatypes http://www.w3.org/TR/xmlschema-2/

WfMC. (2005). XML Process Definition Language (XPDL). http://www.wfmc.org/standards/XPDL.htm

KEY TERMS

Architectural view: a view is a representation of a whole system from the perspective of a related set of concerns (IEEE, 2000)

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

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

Business process modeling: Business Process Modeling (BPM) is the representation of current ("as is") and proposed ("to be") enterprise processes, so that they may be compared and contrasted. By comparing and contrasting current and proposed enterprise processes business analysts and managers can identify specific process transformations that can result in quantifiable improvements to their businesses (Business Process Modeling Forum).

Model-driven Software Development (MDSD) or *Model-driven Development (*MDD*)*: a paradigm that advocates the concept of models, that is, models will be the most important development artifacts at the centre of developers' attention. In MDSD, domain-specific languages are often used to create models that capture domain abstraction, express application structure or behavior in an efficient and domain-specific way. These models are subsequently transformed into executable code by a sequence of model transformations (Völter and Stahl, 2006).

Model and meta-model: a model is an abstract representation of a system's structure, function or behavior. A meta-model defines the basic constructs that may occur in a concrete model. Meta-models and models have a class-instance relationship: each model is an instance of a meta-model (Völter and Stahl, 2006)

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

Role-based Access Control (RBAC): Access control decisions are often based on the roles individual users take on as part of an organization. A role describes a set of transactions that a user or set of users can perform within the context of an organization. RBAC provide a means of naming and describing relationships between individuals and rights, providing a method of meeting the secure processing needs of many commercial and civilian government organizations (Ferraiolo et al., 1999).

Web Service Description Language (WSDL): a standard XML-based language for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate (W3C, 2001)

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

EXERCISES

For the exercises completing this chapter, we are using the following scenario:

At a rescue center rescue missions are controlled. Each emergency call is answered by a co-coordinating officer and is recorded by the control center system. If not supplied by the caller, the officer asks for the following information:

- What happened?
- Who is calling? How can the caller be contacted?

- Where did the accident happen?
- How many people are injured?

After the call, the officer assigns a rescue team to the mission and sends a short description together with the location via, for instance, a Short Data Service (SDS). The rescue team confirms acceptance of the mission by sending a status code '2'. At arrival it notifies the rescue center with status code '3'. After first aid measures, the team prepares to make the patient transportable. When leaving the location the status code is updated to '4'. At the arrival at the hospital with further medical treatment the status is set to '5'. After the team has prepared for standby the rescue center is notified with a status '6'.

Beginner

- Describe the human task of receiving an emergency call. What are the in- and outputs and who may and who may, for example, not perform this task? Define some human roles and describe the relations between them and human tasks as well as the process.
- Table 1 lists basic patterns for control flow modeling that have been defined in the Control-flow View meta-model of the VbMF. UML activity diagrams (OMG, 2004) or Petri-nets (Aalst et al., 2000) are approaches to model process control flows. Transform the textual description of the rescue mission into a UML activity diagram for representing and visualizing the corresponding workflow.

Intermediate

- During the rescue mission multiple participants are involved. BPMN diagrams can help to distinguish these using pools and lanes that represent responsibilities for activities. Identify the different participants that are involved in the rescue mission and draw a BPMN diagram for the rescue mission where you group the process elements that are associated with a participant accordingly.
- Improve the process and provide means for also alerting a fire brigade if necessary. For close collaboration the process itself invokes an external activity by passing the information of the rescue operation to the alarm service of the fire brigade.

Advanced

- A company wants to optimize one of its business workflows. Therefore out of a process with about twenty elements a sub-process containing five process elements is being out-sourced. How do the process models change? Using the view-based approach, what views do you need to modify and where do you need to specify additional information?

Practical Exercise

- BPEL is specified on top of WSDL and XSD. Therefore the conceptual views of the VbMF need to be bound to appropriate syntax. For the Information View e.g. the messages that are being sent have to be defined in XML schemata. For the example of the rescue mission specify XML schemata for the messages that are being sent and extend them with chronological information.