View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA

Huy Tran and Uwe Zdun and Schahram Dustdar

Information System Institute
Vienna University of Technology, Austria
htran,zdun,dustdar@infosys.tuwien.ac.at

Abstract. In process-driven, service-oriented architectures (SOA), process activities invoke services to perform the various tasks of the process. As the number of elements involved in a business process architecture, such as processes, process activities, and services, grows, the complexity of process development also increases along with the number of the elements' relationships, interactions, and data exchanges – and quickly becomes hardly manageable. In addition, process-driven SOA models address different stakeholders, such as business experts and technical experts, who require different kinds of information for their work. Finally, process-driven SOA models must deal with constant changes – both at the business level (e.g. business concept changes) and the technical level (e.g. technologies and platform changes). Separation of concerns is a promising approach to manage such development complexity. In this paper, we propose a view-based, model-driven approach with three major contributions: firstly, it captures different perspectives of a business process model in separate, (semi-)formalized views; secondly, it separates different abstraction levels in a business process architecture; thirdly, an extensible model-driven approach to integrate the different view models and abstraction levels is presented. Our approach is beneficial not only in reducing the process development complexity, but also in coping with dynamic changes at all abstraction levels.

1 Introduction

Service-oriented computing is an emerging paradigm that made an important shift from traditional tightly coupled, hard-to-adapt software development to more platform neutral, loosely coupled software development. The interoperable and platform independent nature of services supports a novel approach to business process development by using processes, running in a process engine, to invoke existing services from their process activities (aka process tasks, steps). We call this kind of architecture *process-driven, service-oriented architecture* [6]. In this approach, a typical business process consists of many activities, the control flow, and process data. Each activity is correspondent to a communication task (e.g., invoking other services, processes, 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 both research and industry, this approach has led to a number of standardization efforts, such as BPEL4WS [7], XPDL [27], BPMN [14], and WS-CDL [26].

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. It is error-prone and time consuming for developers to work with large business processes that implement 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, etc. In addition, this problem also occurs at different abstraction levels [6]. For instance, the business process is relevant for different stakeholders, and business experts require a high-level business-oriented understanding of the various process elements (e.g., relation 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).

In addition to this 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 (technologies, platforms, etc.) constantly evolves.

One of the successful approaches to manage complexity is *separation of concerns* [5]. Process-driven SOAs use a specific realization of this principle, *modularization* [5]: 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, but from the modelers' point of view this 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 *concerns* [8]. 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.

Inspired by the concept of architectural views, we suggest a view-based approach to modeling of process-driven SOAs. Namely, perspectives on business process models and service interactions – as the most important concerns in process-driven SOA – are used as central views in our approach. The approach is extensible with all kinds of other views. In particular, our approach offers separated views, in which each of them represents a certain part of the processes

and services such as control view, interaction view, information view, etc. These views can be viewed separately to get a better understanding of a specific concern, or they can be integrated to produce a richer view or a thorough view of the processes and services.

Technically, our concepts are realized using the model-driven software development (MDSD) paradigm [23][1]. We have chosen this approach to integrate the various view models into one model, and to automatically generate platform-specific or executable code in WSDL, BPEL, or Java. MDSD is also used to separate these platform-specific views from the platform-neutral views and the 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 view models or integrated models into executable code.

The paper is organized as follows. We first provide an overview of the proposed modeling framework and some basic concepts in Section 2. Then, Section 3 gives deeper insight into the modeling framework, followed by a discussion of the extension-, integration-, and transformation-mechanisms. In Section 4, a simple case study, namely, a *Shopping* process, is used to illustrate the realization of the modeling framework concepts. Our related work is discussed in Section 5. Finally, we summarize the main points of the paper, and broaden the research with some outlooks.

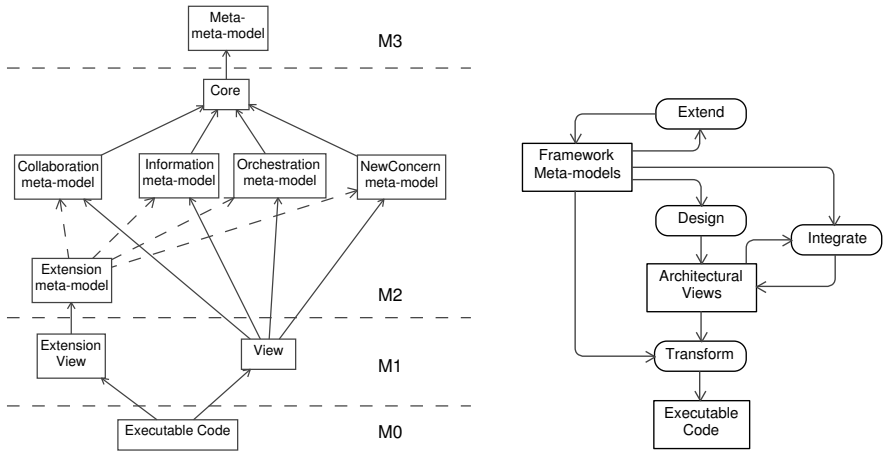
2 Overview of the modeling framework

In this section, we briefly introduce our view-based modeling framework. The framework consists of modeling elements such as a *meta-meta-model*, *meta-models*, and *views* (see Figure 1(a)). As mentioned in the previous section, a view is a representation of a process from the perspective of related *concerns*. In our framework, a view is specified using an adequate framework's meta-model. Each meta-model is a (semi-)formalized representation of a particular business process concern. Therefore, the meta-model specifies entities and their relationships that can appear in the correspondent view. The meta-models, in turn, are defined on top of the *meta-meta-model*. The meta-meta-model can be simple or more elaborate like MOF. Figure 2(a) shows the relevant excerpt of the meta-meta-model of the Eclipse Modeling Framework [3] (i.e., Ecore meta-model) that we used to define our meta-models.

In our approach we categorize distinct activities – in which the modeling elements are manipulated (see Figure 1(b)):

- *Design* is used to define new architectural views.
- *Extend* is used to create a new meta-model by adding more features to an existing meta-model, or by developing it from scratch (e.g., to add a new formalization of a certain business process concern to the framework).
- *Integrate* is used to combine views to produce a richer view or a thorough view of a business process.

[1] Please note that the OMG's MDA proposal is one specific MDSD approach.



(a) Modeling elements (b) Framework overview

Fig. 1. View-based model-driven framework

– *Transform* is used to generate executable code from one or many architectural views.

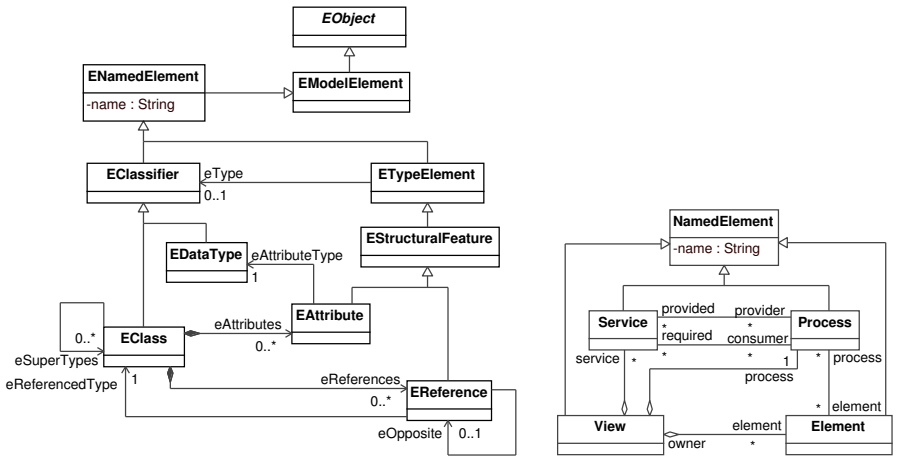
Before generating outputs, *Transform* and *Integrate* validate the input views against relevant meta-models. *Extend* and *Integrate* are the most important activities used to broaden our view-based model-driven framework toward various dimensions. Existing meta-models can be enhanced using the *extension mechanisms* provided in Section 3.5, or can be combined using the meta-model-level *integration mechanisms* provided in Section 3.6.

3 View-based modeling framework

A business process often contains various concerns that require support of modeling approaches. In this paper we firstly concentrate on modeling of the basic concerns of a business process, namely, *orchestration*, *information*, and *collaboration* (see Figure 1(a)). However, our view-based modeling framework is not only bound to the above-mentioned concerns but also open and extensible to allow other concerns such as transactions, event handling, security, quality of service, etc., to be plugged in using the same approach. In the next sections, we present in detail (semi-)formalized representations of the process's concerns summarized above in terms of relevant meta-models along with the discussion of extensibility mechanisms, namely, *extension* and *integration*.

3.1 The Core meta-model

To enhance the extensibility, we devise a basic meta-model, namely, the *Core* meta-model as a foundation for the other meta-models (see Figure 2(b)). Each of



(a) Meta-meta-model excerpt (b) The core meta-model

Fig. 2. Meta-meta-model and the *Core* meta-model

the other meta-models is defined by extending the Core meta-model. Therefore, the meta-models are independent of each other. The Core meta-model is the place where the relationships among the meta-models are maintained. Accordingly, the relationships in the Core meta-model are needed for both view- and meta-model-level integrations as described in Section 3.6.

The Core meta-model consists of a number of abstract meta-classes such as *View*, *Process*, *Service*, and *Element*. These entities are cornerstones of our modeling framework. Each of them can be extended further. At the heart of the *Core* meta-model is the *View* meta-class that captures the central view concept. Each specific view (i.e. each instance of the View meta-class) represents a perspective on one *Process*. It consists of a number of *Services* representing the external functions the business process provides or requires, and a number of *Elements* representing the objects that appear inside the process. Because the meta-models represent concerns of a business process, they are mostly derived from the core meta-model, and the *Service* and *Element* meta-classes are the most important extension points. Moreover, the hierarchical structures in which those meta-classes are roots can be used to define the *integration points* used to combine meta-models (see Section 3.6).

3.2 Orchestration view meta-model

Orchestration is one of the most important concerns of a SOA process. An orchestration 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 orchestration view is specified based on the orchestration view meta-model.

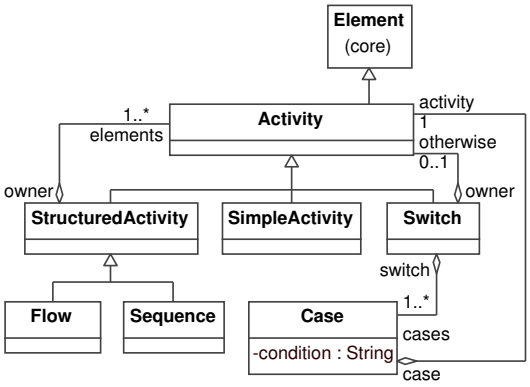


Fig. 3. Orchestration view meta-model

There are several approaches to modeling process's orchestration such as state-charts, block structures [7], activity diagrams [13], Petri-nets [20], and so on. Despite of this diversity in control flow modeling, it is well accepted that existing modeling languages share five basic patterns: *sequence*, *parallel split*, *synchronization*, *exclusive choice*, and *simple merge* [21,22,28]. Thus, we adopted these patterns as the building blocks of our orchestration meta-model. Other, more advanced patterns can be added later by using *extension mechanisms* given in Section 3.5 to augment the orchestration model.

The control structures of BPEL [7], such as *sequence*, *flow*, and *switch*, are more or less equivalent to the aforementioned patterns. The issue here is that the semantics of BPEL's structures is not as clear and precise as the semantics of the patterns. Therefore, instead of re-inventing a new orchestration meta-model we built our meta-model on the basic BPEL control structures, and define their semantics more strictly (see Table 1).

The primary entity of the orchestration meta-model is the *Activity* meta-class (see Figure 3) which is the base class for other meta-classes such as *Sequence*, *Flow*, and *Switch*. Another important entity in the orchestration meta-model is the *SimpleActivity* meta-class that represents a concrete action such as a service invocation, a data processing task, etc. The actual description of each *Simple-Activity* 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 Simple-Activity becomes an integration point to combine an orchestration view with an information view, or with a collaboration view (see *integration mechanisms* in Section 3.6).

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. Semantic of control structures

3.3 Collaboration view meta-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 exposed in terms of standard interfaces (e.g., WSDL portTypes). We captured these concepts in the *Core* meta-model by the relationships between the two elements *Process* and *Service*. The collaboration view meta-model extends the Core meta-model to represent the interactions between the business process and its partners.

In the collaboration view meta-model, the *Service* meta-class from the *Core* meta-model is extended by a tailored *Service* meta-class 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. Therefore, a *Channel* holds a reference to a *Message* entity. Each Message becomes an *integration point*, that can be used to combine a specific collaboration view with an information view (see Section 3.6).

The ability and the responsibility of an interaction partner are modeled by the *Role* meta-class. Every partner – who provides the relevant interface associated with a particular role – can play that role. An interaction between the process and any partner is represented by the *Interaction* meta-class that associates with a specific *Role* of that partner.

3.4 Information view meta-model

The third basic concern we considered in modeling a business process is *information*. This concern is (semi-)formalized by the *information view meta-model*

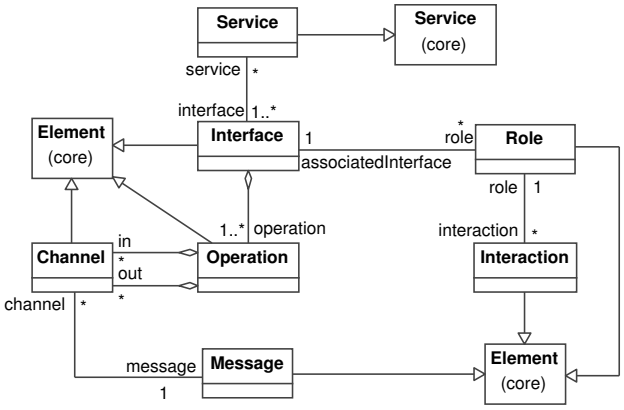


Fig. 4. Collaboration view meta-model

(see Figure 5). This meta-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 meta-model, the *BusinessObject* meta-class, which has the type *ObjectType*, 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, etc. Each piece of information might be a *SimpleBusinessObject*, or a *ComplexBusinessObject* that consists of a number of *BusinessObjects*. We define the *BusinessObjectPool* meta-class as a generic container for 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 transformation is an *ObjectReference* entity that holds a reference to a certain BusinessObject.

3.5 Extension mechanisms

The aforementioned meta-models are the cornerstones to create architectural views like orchestration-, collaboration-, and information-views. Our framework is not limited to these concerns but it allows other concerns to be plugged in via *extension points*. An extension point is any entity that can add additional features (e.g., attributes or relations) to construct a new entity. Using relationships, such as *generalization*, *extend*, etc., we can gradually *refine* an existing meta-model toward another meta-model at a lower abstraction level. For instance, the *orchestration view*, *collaboration view*, and *information view* meta-models are mostly extensions of the Core meta-model using the generalization relation. We also demonstrate the extensibility of the collaboration view meta-model by

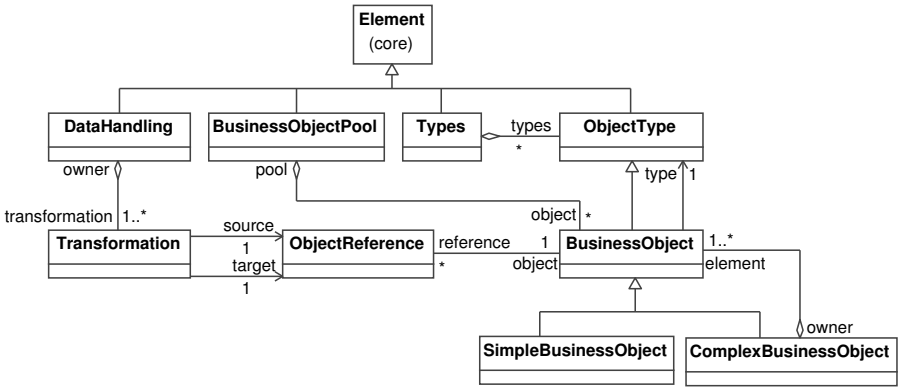


Fig. 5. Information view meta-model

an enhanced meta-model, namely, the *BPELCollaboration* extension (see Figure 6). In the same way, more specific meta-models for other technologies can be derived. In addition, any other business process concern, such as transactions, event handling, and so on, can be (semi-)formalized by a new meta-model derived from the common meta-meta-model using the same approach as used above.

3.6 Integration mechanisms

In our approach, the orchestration 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 (see Algorithm 1).

Definition 1. *Let M_1, M_2 be two meta-models based on a common meta-meta-model. If the entities $m_1 \in M1$ and $m_2 \in M2$ extend the same entity of the meta-meta-model, m_1 and m_2 are* ***conformable****.*

Definition 2. *Given M_1, M_2 are two meta-models and V_1, V_2 are two views conforming to M_1 and M_2, respectively. An* ***integration point*** *between V_1 and V_2 is a tuple $I(v_1, v_2 | v_1 \in V_1,\ v_2 \in V_2,\ v_1 = instanceOf(m_1),\ v_2 = instanceOf(m_2))$, and m_1 and m_2 are conformable, such that V_1 can be merged with V_2 – at the position of v_2 into that of v_1.*

The *GetIntegrationPoint* function receives as input an entity $v_1 \in V_1$ and a view V_2. It looks for $v_2 \in V_2$ such that (v_1,v_2) is an *integration point* between V_1 and V_2. This function can be implemented based on named-based matching, class hierarchical structures, or ontology-based structures. The named-based matching mechanism might be 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

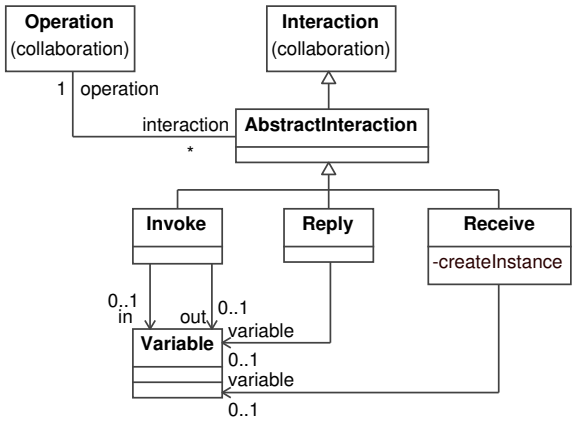


Fig. 6. An extension of the collaboration view

Algorithm 1: View integration algorithm

```
Input: View V1, view V2
begin
    foreach Entity v1 ∈ V1 do
        v2 = GetIntegrationPoint(v1, V2);
        if ( v2 != NULL) then
            v1.add(v2.eAttributes);
            v1.add(v2.eReferences);
        end
    end
end
```

name to the modeling entities which pose the same functionality and semantics. To demonstrate the view integration idea, we present a simple implementation of the name-based matching mechanism (Algorithm 2) for the *GetIntegrationPoint* function.

To create an integrated view – as the result of view integration – a correspondent meta-model of the view has to be defined first. That meta-model is also used later to validate or transform the integrated view into code. Therefore, an adequate integration at the meta-level is needed for any view integration or integrated view transformation. We can use the same approach as used for view integration. However, at the meta-model level, name-based matching is not sufficient. The reason is that the relationships between meta-classes are mostly hierarchical, and the meta-classes that have the same name might not be *conformable*. Therefore, class hierarchical structures are used at the meta-level to define the integration points in our framework. We proposed the *meta-level integration mechanism* using the class hierarchical relationship to define the *meta-level integration points*.

Algorithm 2: Named-matching algorithm

```
Input: Entity v1 ∈ V1, view V2
Output: Entity v2 ∈ V2 or NULL
begin
    Found = FALSE;
    while NOT Found do
        v2 = getNextEntity(V2);
        if v2.name == v1.name then Found = TRUE
    end
    if Found then return v2 else return NULL
end
```

Definition 3. *Given M_1, M_2 are two meta-models based on a common meta-meta-model. A tuple $MI(m_1, m_2 | m_1 \in M_1, m_2 \in M_2)$ is a* ***meta-level integration point*** *iff m_1 and m_2 are conformable and M_1 can be integrated with M_2 by merging the model structure at the position of m_2 into that of m_1.*

3.7 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 our 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, etc. In the literature there are numerous code generation techniques such as templates+filtering, template+meta-model, inline generation, code weaving, etc. [23]. In our prototype, we used the template+meta-model technique – which is realized in the openArchitectureWare framework (oAW) [15] to implement the model transformations. But any of above-mentioned techniques can be utilized in our framework with reasonable modifications.

4 Case study

To demonstrate the realization of the aforementioned concepts, we explain a simple but realistic case study, namely, a *Shopping* process (see Figure 7). The BPEL syntax is adopted to model the Shopping process, and the graphical notations are borrowed from the Eclipse BPEL Designer environment [4].

In the next paragraphs, we present an illustrative case study by the following steps. Firstly, architectural views of the Shopping process are designed based-on our meta-models and the sample extension for BPEL constructs, given in Figure 6. Secondly, some views are integrated to produce a richer perspective. And finally, these views are used to generate executable code in BPEL4WS [7] and WSDL [24] that can be deployed into any BPEL engine.

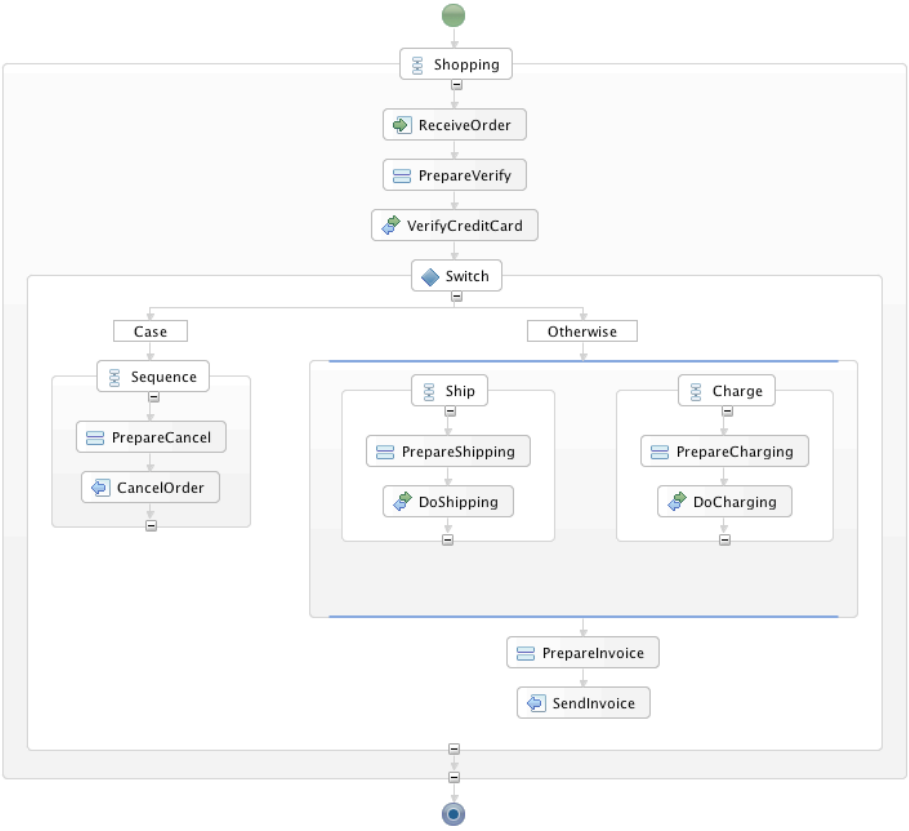


Fig. 7. The shopping case study

4.1 The Shopping process

The Shopping process is initiated when the process's customer issues a purchase order. The purchase order is retrieved via the *ReceiveOrder* activity. The process then invokes 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 *PrepareVerify* that extracts these information from the purchase order. The preparation step is executed before an interaction on the process takes place in order to arrange the needed input data for the interaction. The control after validating the customer's credit card is divided into two branches according to the validation results. 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 response along with an explaining message. Otherwise, the positive confirmation will trigger the second control branch in which the process continues with two concurrent activities,

Fig. 8. Shopping process orchestration view

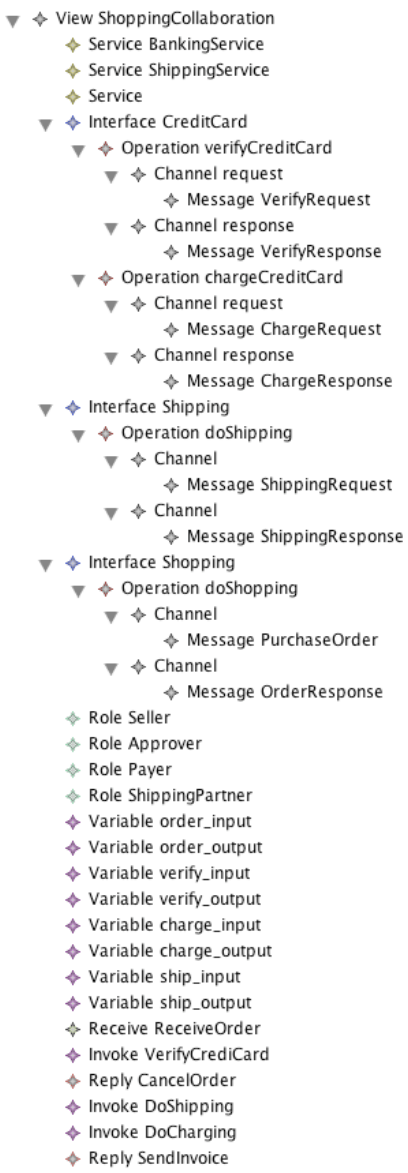
DoShipping and *DoCharging*. *DoShipping* gets shipping information from the purchase order and delivers ordered products to the customer, while *DoCharging* sends a request to the Banking service for the credit card's payment. Finally, the purchase invoice is prepared and sent back to the customer during the last step, *SendInvoice*. After that, the Shopping process successfully finishes.

4.2 View development

Figure 8 shows the orchestration model of the Shopping Process. There are no details of data exchanges or service communication in this view. Hence, this view can be used at the business level to capture the business expert knowledge. Because the orchestration view meta-model is based on the BPEL control model excerpt, the structure of the Shopping's orchestration view is quite similar to that in Figure 7.

Moreover, using the extension meta-models (e.g., see Figure 6) we 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 information model (see Figure 9(a)) and another one is a view based on the *BPELCollaboration* meta-model (see Figure 9(b)).

(a) Shopping collaboration view



(b) Extension view using BPELCollaboration

Fig. 9. View and extension view of Shopping process

4.3 View integration

The views also can be integrated to produce new richer views of the Shopping process. In Figure 10, the collaboration view of the Shopping process (see Fig-

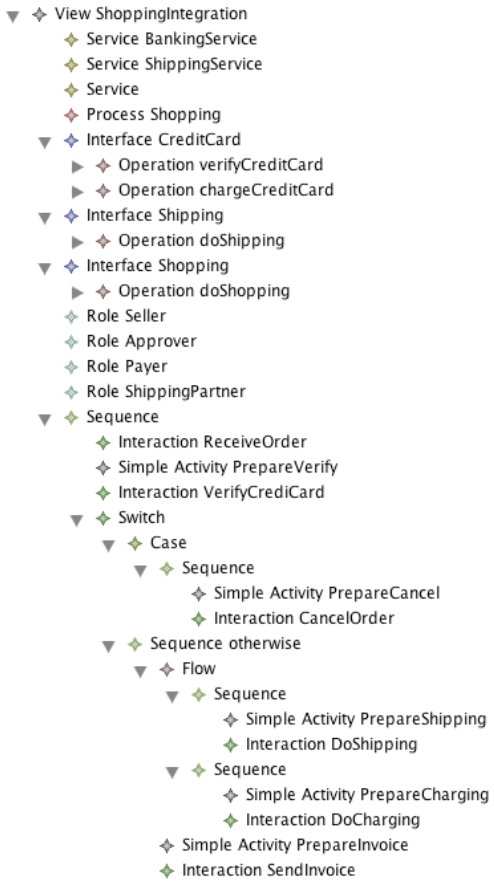


Fig. 10. Integration of orchestration and collaboration views

ure 9(a)) is integrated with the orchestration view (see Figure 8). The most important integration points are defined by *SimpleActivity* in the orchestration view with relevant *Interaction* entities in the collaboration view. The output view consists of the control structures based on the orchestration view with other collaboration-related entities such as Role, InteractiveServices, etc.

4.4 Code generation

After modeling the Shopping process, we developed 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 Ecore models. We used the oAW's Xpand language [15] to define our model transformations. Figure 11 shows a transformation snippet in oAW's

```
#############################################################################################
# Template for the SimpleActivity of an OrchestrationView which uses named based matching #
# to integrate with an Interaction entity in a CollaborationView, or a DataHandling       #
# entity in an InformationView                                                              #
#############################################################################################
«DEFINE Activity FOR orchestration::SimpleActivity-»
    «EXPAND SimpleActivity FOR getActivityByName(this.name)-»
«ENDDEFINE»

«DEFINE SimpleActivity FOR bpel_collaboration::Invoke-»
    <invoke name="«name»" partnerLink="«getPartnerLink()»"
        outputVariable="«getOutput()»" inputVariable="«getInput()»"
        portType="«getInterface()»" operation="«getOperation()»"/>
«ENDDEFINE»

«DEFINE SimpleActivity FOR bpel_collaboration::Receive-»
    <receive name="«name»" partnerLink="«getPartnerLink()»" variable="«getVariable()»"
        operation="«getOperation()»" portType="«getInterface()»"
        «IF createInstance == true»
        createInstance="yes"
        «ENDIF»/>
«ENDDEFINE»

«DEFINE SimpleActivity FOR bpel_collaboration::Reply-»
    <reply name="«name»" partnerLink="«getPartnerLink()»" operation="«getOperation()»"
        variable="«getVariable()»" portType="«getInterface()»" />
«ENDDEFINE»

«DEFINE SimpleActivity FOR bpel_information::Assign-»
    <assign name="«name»">
        ...
    </assign>
«ENDDEFINE»
```

Fig. 11. Code generation for SimpleActivity entities

Xpand language [15] that generates BPEL activities such as *Invoke*, *Receive*, etc. using the extension view in Figure 9(b). The resulting executable code in BPEL and WSDL are successfully deployed on the ActiveBPEL engine [1] as a running illustrative example for the realization of our concepts.

5 Related work

Our work is closely related to existing process modeling languages. There are several standardization efforts for process modeling languages, such as BPEL4WS [7], BPMN [14], XPDL [27], WSCI [25], WS-CDL [26], and so on. They can be categorized into different dimensions, for instance, textual and graphical languages, or abstract and executable languages, and so on. The abstract modeling languages (e.g., abstract BPEL, or WSCI/WS-CDL) are working at the same abstraction level as our abstract models (i.e., orchestration, information, or collaboration models) while the executable language are more or less similar to our refined models. The aforementioned modeling languages consider the business process model as a whole. They do not support the separation of the process model's concerns. Moreover, there is no explicit relationship between an abstract and an executable modeling language. So it requires additional effort to maintain

the integrity and consistency of the models, or to validate models [11, 16]. All these modeling languages can be integrated into our approach using extension models.

To the best of our knowledge, there is only a few view-based approaches to business process modeling. The most related work in this area is the approach by Mendling et al. [12] inspired by the idea of schema integration in database design. Process models based on Event-driven Process Chains (EPCs) are investigated, and the pre-defined semantic relationships between model elements such as *equivalent*, *sequence*, and merge operations are performed to integrate two distinct views. Semantics-based merging is a promising approach to model integration, but it is difficult to apply to integrate two different types of models, for instance, to merge a control model with a data model. Thus, the authors mainly focus on integrating process models without any data element or any collaboration.

The Amfibia [2,10] approach focuses on formalizing different aspects of business process modeling, and/or develop an open framework to integrate various modeling formalisms through the *interface* concept. Akin to our approach, 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. Like our approach, Amfibia's framework also has a core model with a small number of important elements, which are referred to, or refined in other models. The distinct point to our framework 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 our framework, the integrity and consistency between models can be verified earlier at the model level.

Also, similar to the Amfibia approach, there is a standardized reference model, namely, ISO Reference Model for Open Distributed Processing or RM-ODP [9]. RM-ODP defines a set of different *view points* such as enterprise, information, computational, engineering, and technology viewpoints. Each viewpoints has its own language and semantics. The consistency among viewpoints is ensured by the common architecture and the *common object model*. These concepts, likewise those in Amfibia and our approach, are defined based on the principle of separation of concerns to help stake-holders thinking from different perspectives in order to manage complexity of distributed applications. The advantage of our approach compare to these approaches is that our view-based model-driven framework does not only separate concerns but also separate levels of abstraction, for instance, business level, and technical level.

Our work also shares some concepts with the approach described in [19]. van der Aalst et al. develop a conceptual SOA-based architecture framework around the idea of modularization. The key concept in [19] is the *component* that is more or less equivalent to our *process* concept, and the relationships between components. The authors emphasized on the separation of activities from data

elements, but do not mention the capability of extending or integrating other concerns that could appear on a business process.

Skogan et al. [18] offer another approach for process-based modeling in UML. A tool-chain is devised to extract and formalize WSDL descriptions using UML models. Service compositions are captured by UML activity diagrams with special stereotypes. Finally, code in executable languages is generated from a composition model. The authors neither consider separation of concerns in service composition nor integration of other concerns except service interfaces and the control flow.

Schmidt et al. [17] proposes an interesting approach to web service transaction modeling. Even though the approach is only considering one concern of a business process model, the paper also mentions the separation of views into layers and maintaining references between various layers. Our work has not yet focused on other concerns, such as transactions, security, etc., but our model-driven framework in general can be extended into these dimension using the approach presented in this paper. Consequently, the transaction model in [17] can be seen as a complement to our work to develop the meta-model for the transaction concerns of the business process.

6 Summary and outlook

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. In this paper, we introduced a view-based framework that (semi-)formally defines various concerns of the process model and uses those (semi-)formalized models to capture a particular perspective of the business process. It not only helps to manage the development complexity by the separation of the processes' concerns, but also to cope with both business and technical changes using the separation of abstraction levels.

This study also raises a number of research questions which are only answered by further work. The modeling framework should be extended with other concerns of the business process such as transactions, security, event handling, etc. In addition, the view integration algorithms can be enhanced by the validation of possible constraints conflicts between various integration points. Finally, an ontology-based structure might be richer and be better suited to improve the integration at the meta-level than the class hierarchical structure.

References

1. Active Endpoints. ActiveBPEL Open Source Engine 2.x. http://www.active-endpoints.com/, 2006.
2. B. Axenath, E. Kindler, and V. Rubin. An open and formalism independent meta-model for business processes. In *Proceedings of the Workshop on Business Process Reference Models*, pages 45–59, 2005.
3. Eclipse. Eclipse Modeling Framework. http://www.eclipse.org/emf/, 2006.

4. Eclipse. WS-BPEL Project 0.2.0. http://www.eclipse.org/bpel/, 2006.
5. C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering.* Prentice Hall, 1991.
6. 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.
7. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. Business Process Execution Language for Web Services (BPEL4WS). ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, 05 2003.
8. IEEE. Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE-std-1471-2000, IEEE, 2000.
9. ISO. Open Distributed Processing Reference Model (IS 10746). http://isotc.iso.org/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm, 1998.
10. E. Kindler, B. Axenath, and V. Rubin. AMFIBIA: A Meta-Model for the Integration of Business Process Modelling Aspects. In *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings, 2006.
11. J. Mendling and M. Hafner. From Inter-organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In *OTM Workshops*, pages 506–515, 2005.
12. J. Mendling and C. Simon. Business Process Design by View Integration. In *Business Process Management Workshops*, volume 4103 of *LNCS*, pages 55–64. Springer, 2006.
13. OMG. Unified Modelling Language 2.0 (UML). http://www.uml.org, 2004.
14. OMG. Business Process Modeling Notation (BPMN). http://www.bpmn.org/Documents/OMG-02-01.pdf, 02 2006.
15. openArchitectureWare.org. openArchitectureWare project. http://www.openarchitectureware.org, 08 2002.
16. C. Ouyang, M. Dumas, A. H. M. ter Hofstede, and W. M. P. van der Aalst. From BPMN Process Models to BPEL Web Services. In *ICWS*, pages 285–292, 2006.
17. B. A. Schmit and S. Dustdar. Model-driven Development of Web Service Transactions. *International Journal Enterprise Modelling and Information Systems Architectures*, 1(1):46–, 10 2005.
18. D. Skogan, R. Grønmo, and I. Solheim. Web Service Composition in UML. In *Enterprise Distributed Object Computing Conference, 2004*, pages 47–57, 2004.
19. W. van der Aalst, M. Beisiegel, K. van Hee, D. König, and C. Stahl. A SOA-Based Architecture Framework. In *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings, 2006.
20. W. van der Aalst, J. Desel, and A. Oberweis, editors. *Business Process Management: Models, Techniques, and Empirical Studies - Lecture Notes in Computer Science*, volume 1806. Springer-Verlag, 2000.
21. W. M. van der Aalst, M. Dumas, A. H. ter Hofstede, and P. Wohed. Pattern Based Analysis of BPMN (and WSCI). Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.
22. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
23. M. Völter and T. Stahl. *Model-Driven Software Development: Technology, Engineering, Management.* Wiley, 2006.
24. W3C. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 03 2001.

25. W3C. Web Service Choreography Interface (WSCI). http://www.w3.org/TR/wsci, 08 2002.
26. W3C. Web Services Choreography Description Language (WSCI). http://www.w3.org/TR/ws-cdl-10, 11 2005.
27. WfMC. XML Process Definition Language (XPDL). http://www.wfmc.org/standards/XPDL.htm, 10 2005.
28. P. Wohed, W. M. van der Aalst, M. Dumas, and A. H. ter Hofstede. Pattern Based Analysis of BPEL4WS. Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.