View-based Integration of Process-driven SOA Models At Various Abstraction Levels

Huy Tran and Uwe Zdun and Schahram Dustdar

Distributed Systems Group, Institute of Information Systems
Vienna University of Technology, Austria
`htran,zdun,dustdar@infosys.tuwien.ac.at`

Abstract. SOA is an emerging architectural style to achieve loosely-coupling and high interoperability of software components and systems by using message exchanges via standard public interfaces. In SOAs, software components are exposed as services and typically coordinated by using processes which enable service invocations from corresponding activities. These processes are described in high-level or low-level modeling languages. The extreme divergence in term of syntax, semantics and levels of abstraction of existing process modeling languages hinders the interoperability and reusability of software components or systems being built upon or relying on such models. In this paper we present a novel approach that provides an automated integration of modeling languages at different abstraction levels using the concept of architectural view. Our approach is realized as a view-based reverse engineering tool-chain in which process descriptions are mapped onto appropriate high-level or low-level views, offered by a view-based modeling framework.

1 Introduction

In a Service-oriented Architecture (SOA), software components are often exposed as services that have standard interfaces and can be invoked by message exchanges. A number of relevant services can be coordinated to achieve a specific business functionality. The integration and interoperability of the software components or systems are accomplished by orchestrating them using a process, which is deployed in a process engine. Each process typically consists of a control flow, a number of service invocations and other activities for data processing, fault and transaction handling, and so on. Processes are often developed using modeling languages such as EPC [4, 13], BPMN [9], UML Activity Diagram extensions [8], BPEL [7] or XPDL [15].

Business analysts usually design processes in high abstraction languages, such as BPMN, EPC, or UML Activity Diagram, and developers implement them using executable languages, such as BPEL/WSDL. An important issue that hinders the interoperability and the reusability of existing process models is the huge divergence of these modeling languages. This issue occurs because there is no explicit link between two modeling languages at the same or different abstraction levels. For instance, developers could not re-use or integrate the
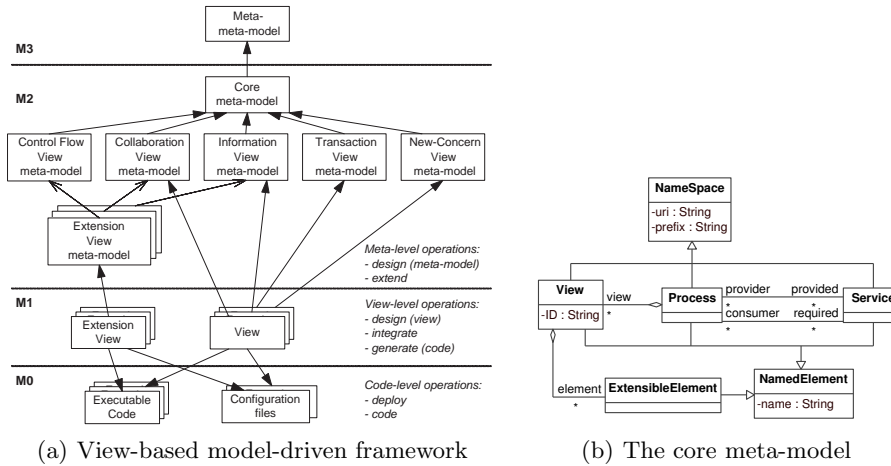
(a) View-based model-driven framework   (b) The core meta-model

Fig. 1. Meta-meta-model and the *Core* meta-model

whole or part of a process described using BPEL in another process developed using BPMN or EPC, and vice versa. The most popular solution for this issue is to define direct transformations between the different process modeling languages [5,6,11,16]. These approaches, even though they partially solve the problem, pose a serious limitation regarding extensibility. First, they focus on one concern, the control flow of the process models, and ignore other crucial concerns, such as collaborations, data processing, fault handling, and so on. Second, each of these transformation approaches only provides the integration of two specific kinds of process models, but provides neither interoperability with process models realized in other languages than those two nor the reusability of these models.

To overcome this issue and the limitations of transformation-based approaches, we present a novel integration approach for enhancing the interoperability and reusability of process-driven SOA models. Our approach exploits the concept of *architectural view* during the bottom-up analysis, then maps the process descriptions onto relevant high-level or low-level views. Using a view-based modeling framework (VbMF), originally introduced in our previous work [12], we can provide the interoperability and reusability between these views, or in other words, between different process modeling languages.

In this paper, we first present an overview of VbMF in Section 2. Section 3 describes the model integration approach we propose in this paper in terms of our view-based reverse engineering tool-chain. Section 4 depicts the empirical analysis of this approach to exemplify process descriptions in popular modeling languages, namely, BPEL and WSDL, via a simple but realistic example. Finally, we compare to related work in Section 5 and conclude.
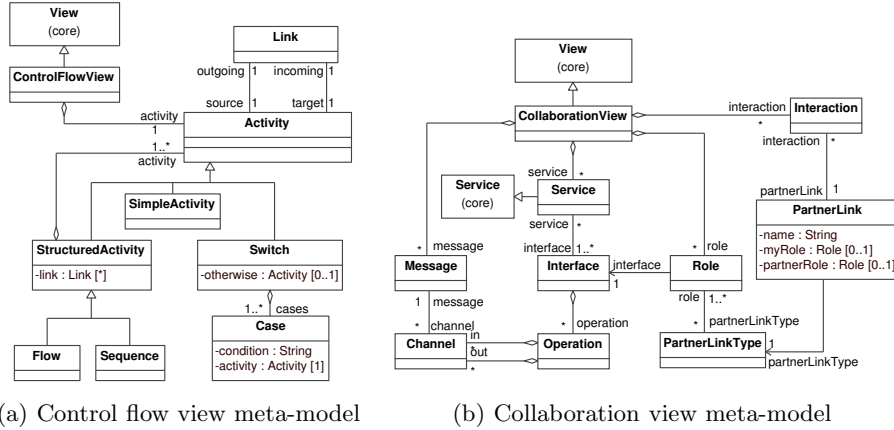
(a) Control flow view meta-model   (b) Collaboration view meta-model

Fig. 2. The control flow view and collaboration view meta-models

2 Overview of View-based Modeling Framework (VbMF)

Figure 1(a) gives an overview of the concepts in VbMF. A view is defined according to a corresponding meta-model, which is a (semi-)formalized representation of a particular process concern. At the heart of VbMF is the Core meta-model (see Figure 1(b)) from which each view meta-model is derived. Example meta-models that we have derived from the Core include the following views [12]: Collaboration (see Figure 2(b)), Control Flow (see Figure 2(a)) and Information. Based on the meta-models, we derive particular views. For specific technologies, for instance, BPEL and WSDL, we provide extension meta-models which comprise additional details required to depict the specifics of these technologies. Figure 3 depicts the BPEL-specific extension of the collaboration view in Figure 2(b). Hence, we can use the distinction of Core meta-model, generic view meta-models, and extension view meta-models to handle different abstraction levels, including business-level concerns and technical concerns.

The main task of the Core meta-model (shown in Figure 1(b)) is to provide integration points for the various meta-models defined as parts of VbMF, as well as extension points for enabling the extension with views for other concerns or more specific view meta-models, such as those for specific technologies. Consider the control flow view meta-model in Figure 2(a) as an example: 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. The control flow view meta-model is defined by extending the View and ExtensibleElement meta-classes from Core.

All other view meta-models are essentially defined in the same way: by extending the model elements provided as integration and extension points in the Core meta-model. The more specific extension view meta-models are also defined in the same way: they extend the elements of their root meta-models and of the
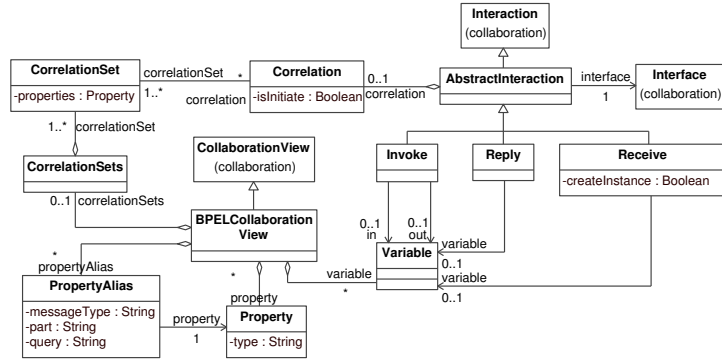
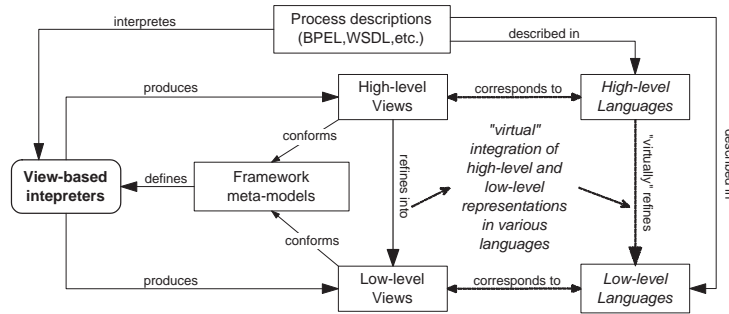**Fig. 3.** BPEL extension of the collaboration view meta-model

Fig. 4. Integration of various modeling languages using view-based reverse engineering

Core meta-model. For instance, a BPEL-specific control flow meta-model with extra elements, such as *while*, *wait*, *terminate*, etc., can be defined by extending the elements from Figures 1(b) and 2(a).

In our implementation of these concepts, we exploit the model-driven software development (MDSD) paradigm [14] to separate the platform-neutral views from the platform-specific views. Platform-specific models or executable code, for instance, Java code, or BPEL and WSDL descriptions, can be generated from the views by using model-to-code transformations. Our prototype is realized using openArchitectureWare (oAW) [10], a model-driven software development tool, and the Eclipse Modeling Framework [3]. The tools allow stake-holders of a process to only view a specific perspective, by examining a single view, or to analyze any combination of views (i.e., to produce an integrated view).

3   View-Based Model Integration

In this section, we present a view-based reverse engineering approach for addressing the divergence issue of modeling languages and for overcoming the lim-

itations of transformation-based solutions. The ultimate goal of this approach is to map the process descriptions onto high-level or low-level views, appropriate for different stakeholders (see Figure 4). To demonstrate our approach, we have exemplified it using the combination of BPEL and WSDL, which are possibly the most popular process and service modeling descriptions used by numerous companies today. The same approach can be taken for any other process-driven SOA technologies. In addition, due to space limitation, we only present the extraction and the integration of two basic views in VbMF: the control flow view and the collaboration view. Other views, for instance, the Information View, the Transaction View, etc., can be integrated using the same approach.

The tool-chain consists of a number of view-based interpreters, such as *control flow interpreter*, *information view interpreter*, *collaboration view interpreter*, and so on. Each interpreter is responsible for extracting one view from the process descriptions. Therefore, an interpreter for a certain view must be defined based on the meta-model which that view conforms to. For instance, the *control flow view* consists of elements, such as *Activity*, *Flow*, *Sequence*, *Switch*, etc. (see also Figure 2(a)). To extract the control flow view from process descriptions, the interpreter walks through the input descriptions to pick only these elements and ignores others. As the modeling framework grows with additional views, the reverse engineering tool-chain can be scaled to fit to the growing framework. To add a new view to the framework, an adequate meta-model of this view has to be specified using extension mechanisms [12]. Next, we develop an interpreter based on the new meta-model specification and hook the it into the tool-chain.

4 Empirical Analysis of Model Integration

In this section, we exemplify our approach for the combination of the process and service modeling languages BPEL and WSDL. However, the concepts are not specific for BPEL/WSDL, but applicable in the same way for other process-driven modeling languages.

To demonstrate the realization of aforementioned concepts, we present a simple but realistic case study, namely, a *Shopping process*, which depicts a typical e-commerce scenario. The Shopping process is represented in BPEL and WSDL. The process starts when the customer's purchase order arrives together with their billing information (e.g., credit card). Then, the process invokes a *Banking service* to validate the customer's credit card through the *VerifyCreditCard* activity. If the validation is successful, the customer will obtain the purchased items which is shipped by a delegated *Shipping service*. After that, the process will send the order invoice to the customer. Otherwise, the order will be canceled as a negative confirmation is received from the Banking service. An excerpt from the BPEL/WSDL code is shown on the left hand side of Figures 5 and 6.

4.1 Control Flow View Mapping

According to the specification of the control flow view meta-model (see Figure 2(a)), a control flow view consists of elements which represent the control
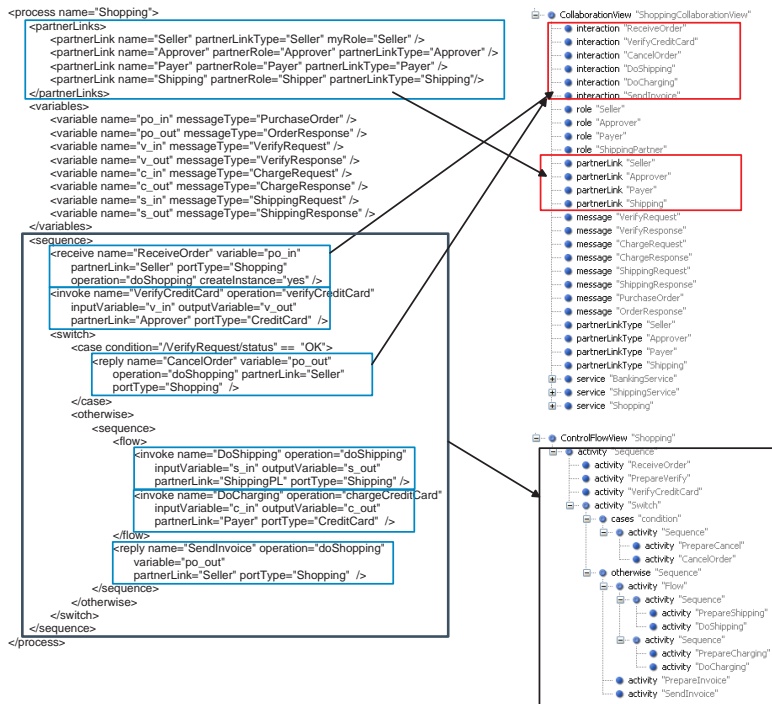
Fig. 5. The mapping of the Shopping process descriptions in BPEL to the collaboration view (top-right) and the control flow view (bottom-right).

flow of a business process. The hierarchy and the execution order of the control flow are defined using basic structured activities. These structured activities include: *sequence*, for defining a sequential execution order; *flow*, for the concurrent executions; and *switch-case-otherwise*, for conditional branches. Structured activities can be nested and combined in arbitrary manners to represent various complex control flows in BPEL processes. In addition, BPEL also has primitive activities, such as *invoke*, an service invocation; *receive*, waiting for a message from partners; *reply*, sending back a response to a certain partner; *assign*, assigning values to BPEL variables.

The interpreter walks through the process description in BPEL and collects the information of atomic and structured activities. Then, it creates the correspondent elements in the view and assigns relevant values (e.g. the *name* attribute) to their attributes. We describe in Table 1 the mapping specification between BPEL and the control flow view elements/attributes. The BPEL mapping is illustrated in Figure 5.

Table 1. The basic mapping from BPEL onto the control flow view

BPEL element	Control flow view element
invoke, receive, reply, assign *name="..."*	controlflow::SimpleActivity *setName()*
sequence *name="..."*	controlflow::Sequence/*setName()*
flow *name="..."*	controlflow::Flow/*setName()*
switch *name="..."*	controlflow::Switch/*setName()*
case *name="..."* *condition="..."*	controlflow::Case *setName()* *setCondition()*
otherwise *name="..."*	controlflow::Otherwise/*setName()*

Table 2. The basic mapping from WSDL onto collaboration view elements

WSDL element	Collaboration view element
definition	core::Service
message *name="..."*	collaboration::Message/*setName()*
portType *name="..."*	collaboration::Interface/*setName()*
operation *name="..."*	collaboration::Operation/*setName()*
input,output *name="..." message="..."*	collaboration::Channel *setName()*, *setMessage()*
plnk::partnerLinkType *name="..."*	collaboration::PartnerLinkType/*setName()*
plnk::Role *name="..."*	collaboration::Role/*setName()*
service *name="..."*	core::Service.*setName()*

4.2 Collaboration View Mapping

The collaboration view interpreter is realized using the same approach as the control flow interpreter. However, the collaboration view comprises not only the elements from BPEL but also from WSDL. Hence, first of all, the interpreter has to collect all service interface, message, role, partnerLinkType descriptions from WSDL. Then, the interpreter creates relevant elements in the collaboration view according the mapping rules given in Table 2. Figure 6 depicts the mapping of the Shopping process in WSDL to corresponding elements in the collaboration view. Next, the interpreter walks through the BPEL code to extract collaborative elements in a similar manner. The basic activities, namely, *invoke*, *receive*, and *reply*, appear on the collaboration view with the same name as in the control flow view. However, these activities contain additional collaborative attributes as depicted in Table 3. The BPEL mapping is illustrated in Figure 5. Besides generating the collaboration view's elements, the interpreter uses the information collected in the former step to establish necessary relationships between these elements. For instance, the relationship between *collaboration::Interaction* and *collaboration::PartnerLink* elements is derived from the association between the communication activities (e.g.,*invoke*, *receive*, *reply*) and the *partnerLink*, or the relationship between *collaboration::PartnerLink* and *collaboration::PartnerLinkType* is derived from the association among the *part-*

Fig. 6. The mapping of the Shopping process descriptions in WSDL to the collaboration view.

nerLinkType elements in WSDL and the *partnerLink* elements in BPEL, and so on.

4.3 BPEL-extension Collaboration View Mapping

According to the collaboration view meta-model, we map the Shopping process onto a high abstraction view as shown in Figures 6 and 5. This view is suitable for communication with business analysts, but it does not provide appropriate information for IT experts. As we mentioned in Section 2, the collaboration view can be refined into a lower abstraction view, namely, the BPELCollaboration View (see Figure 3). To demonstrate the concept of view refinement, we present the mapping of the Shopping process in BPEL to the corresponding BPEL-specific collaboration view in Figure 7 according to the specification in Table 3. For the sake of readability, we omit the elements inherited from the collaboration view and depict an excerpt of the BPEL-extension collaboration view together with additional features.

4.4 Discussion

In the previous empirical analysis, we illustrated the mapping of process descriptions onto high-level or low-level views. The high-level views in our approach are platform-independent models designed to capture abstract features in a process.
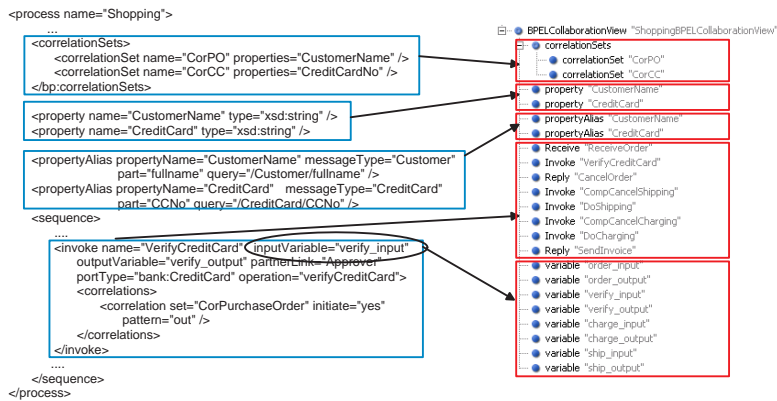
Fig. 7. The mapping of the Shopping process descriptions in BPEL to the BPEL collaboration view

Corresponding to these views are high-level modeling languages, such as EPC, BPMN or UML Activity Diagram. In this paper, we illustrated a mapping of such models into our framework from high-level view models, which reflect the concepts in the low-level models rather closely. This has a number of advantages. Firstly, it uses one kind of modeling approach for all types of views. Secondly, it avoids any semantic mismatch or transformation between modeling concepts. But, on the other hand, this approach has the disadvantage that existing modeling language code (say realized in EPCs, BPMN or UML Activity Diagrams) would have to be mapped to our high-level models, which could be a considerable effort for huge existing process repositories. But in general this is possible and can even be largely automated, because our control flow view represents five basics patterns that exist in any modeling language; the collaboration view describes generic interactions that typically occur between a process and its partners [12]. Hence, an adequate interpreter for a certain language can distill these views from the process descriptions in that language using our approach. Alternatively, our approach can of course also be extended with a respective new view model, such as an EPC or BPMN control flow view.

A high-level view can be refined into a low-level view which captures the specifics of a particular technology. For example, the collaboration view is extended and refined to the BPEL collaboration view that embodies several BPEL-specific features. Using the same approach, we can define appropriate meta-models and interpreters for pulling out corresponding views from process implementations in low-level, executable languages such as BPEL. Using VbMF the stakeholders can work on a particular view or can examine any combination of several views instead of manipulating various kinds of process descriptions or digging into the implementation code in executable languages. Our approach can help the stakeholders to quickly understand and grasp the information in (different) modeling languages as well as re-use adequate views.

5 Related Work

In the software engineering area, the concept of *reverse engineering* is the process of analyzing a system to *identify the system's components and their relationships* and *create representations of the system in another form or at a higher level of abstraction* [1, 2]. We devised a novel view-based reverse engineering approach that supports extracting relevant abstraction levels of process representations in terms of architectural views. Various modeling languages can be integrated into VbMF and manipulated or re-used in other process models.

Existing modeling languages provide high-level abstractions, such as EPC [4,13], BPMN [9], or UML Activity Diagram extensions [8], or low-level and executable descriptions, such as BPEL [7], or XPDL [15]. There are several efforts to transform process models described in one language into models represented in another language. For instance, Mendling et al. [6] present the mapping of BPEL to EPCs; Ziemann et al. [16] report on an approach to model BPEL processes using EPC-based models; Recker et al. [11] translate between BPMN and BPEL; Mendling et al. [5] discuss X-to-BPEL and BPEL-to-Y transformations. These transformation-based approaches mostly focus on one concern of the process models, namely, the control flow. There is no support for handling or integrating other process concerns, such as service interactions, data processing, or transaction handling. Moreover, each of these approaches only provide the integration of a certain pair of process modeling languages, but does not offer the interoperability of process models in other languages, or the reusability of these models to develop other processes.

Zou et al. [17] propose an approach for extracting business logic, in term of workflows, from existing e-commerce applications. The analyzing process is guided by documented workflows to identify the business logic. Then, the business logic is captured in terms of the control flows using the concept of process algebra. This approach aims at providing high-level representations of processes and maintaining the relationships among different abstraction levels to quickly re-act to changes in business requirements. This approach only focuses on control flow and does not target other concerns. In addition, there is no support for the interoperability and the re-usability of different software components.

6 Conclusion

Interoperability and reusability suffer from the heterogeneous nature of the participants of a software system. SOA partially reconciles this heterogeneity by defining standard service interfaces as well as messaging mechanisms for communicating between services. Process-driven SOAs provide an efficient way of coordinating various services in terms of processes to accomplish a specific business goal. However, the huge divergence of process modeling languages raises a critical issue that deteriorates the interoperability and the reusability of software components or systems. Our approach, presented in this paper, exploits the concept of architectural views and a reverse engineering tool-chain to map high-level

or low-level descriptions of processes into appropriate views. The resulting views are integrated into the view-based modeling framework and can be manipulated or re-used to develop other processes.

References

1. T. J. Biggerstaff. Design recovery for maintenance and reuse. *IEEE Computer*, 22(7):36–49, 1989.
2. E. J. Chikofsky and J. H. I. Cross. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, 1990.
3. Eclipse. Eclipse Modeling Framework. http://www.eclipse.org/emf/, 2006.
4. E. Kindler. On the semantics of EPCs: A framework for resolving the vicious circle. In *Business Process Management*, pages 82–97, 2004.
5. J. Mendling, K. B. Lassen, and U. Zdun. Transformation strategies between block-oriented and graph-oriented process modelling languages. Technical Report JM-200510 -10, WU Vienna, 2005.
6. J. Mendling and J. Ziemann. Transformation of BPEL processes to EPCs. In *Proc. of the 4th GI Workshop on Event-Driven Process Chains (EPK 2005)*, volume 167, pages 41–53, Dec 2005.
7. OASIS. Business Process Execution Language (WSBPEL) 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf, May 2007.
8. OMG. Unified Modelling Language 2.0 (UML). http://www.uml.org, 2004.
9. OMG. Business Process Modeling Notation. http://www.bpmn.org/Documents/OMG-02-01.pdf, Feb 2006.
10. openArchitectureWare.org. http://www.openarchitectureware.org, Aug 2002.
11. J. Recker and J. Mendling. On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In *Eleventh Int. Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'06)*, pages 521–532, Jun 2006.
12. H. Tran, U. Zdun, and S. Dustdar. View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA. In *Intl. Working Conf. on Business Process and Services Computing (BPSC'07)*, volume 116 of *Lecture Notes in Informatics*, pages 105–124, sep 2007.
13. W. van der Aalst. On the verification of interorganizational workflows. Computing Science Reports 97/16, Eindhoven University of Technology, 1997.
14. M. Völter and T. Stahl. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 2006.
15. WfMC. XML Process Definition Language (XPDL). http://www.wfmc.org/standards/XPDL.htm, Apr 2005.
16. J. Ziemann and J. Mendling. EPC-based modelling of BPEL processes: a pragmatic transformation approach. In *Proc. of the 7th Int. Conference "Modern Information Technology in the Innovation Processes of the Industrial Enterprises" (MITIP 2005)*, 2005.
17. Y. Zou and M. Hung. An approach for extracting workflows from e-commerce applications. In *ICPC '06: Proc. of the 14th IEEE Int. Conf. on Program Comprehension (ICPC'06)*, pages 127–136, Washington, DC, USA, 2006. IEEE Computer Society.

Table 3. The basic mapping from BPEL description onto the collaboration view

BPEL element	Collaboration view element	BPELCollaboration view element
invoke *name="..."* *partnerLink="..."* *portType="..."* *operation="..."* *inputVariable="..."* *outputVariable="..."* *correlation set="..."*	collaboration::Interaction *setName()* *setPartnerLink()* *setInterface()*	bpel::Invoke *setName()* *setPartnerLink()* *setInterface()* *setOperation()* *setInput()* *setOutput()* *createCorrelation()*
receive/reply *name="..."* *partnerLink="..."* *portType="..."* *operation="..."* *variable="..."* *createInstance="yes"* *correlation set="..."*	collaboration::Interaction *setName()* *setPartnerLink()* *setInterface()* *setCreateInstance(true)*	bpel::Receive/bpel::Reply *setName()* *setPartnerLink()* *setInterface()* *setOperation()* *setVariable()* *setCreateInstance(true)* *createCorrelation()*
partnerLink *name="..."* *partnerLinkType="..."* *myRole="..."* *partnerRole="..."*	collaboration::PartnerLink *setName()* *setPartnerLinkType()* *setMyRole()* *setPartnerRole()*	*(inherits from parent – the collaboration view)*
correlationSets		bpel::CorrelationSets
correlationSet *name="..."* *properties="..."*		bpel::CorrelationSet *setName()* *setProperty()*
property *name="..."* *type="..."*		bpel::Property *setName()* *setType()*
propertyAlias *propertyName="..."* *messageType="..."* *part="..."* *query="..."*		bpel::PropertyAlias *setProperty()* *setMessageType()* *setPart()* *setQuery()*